The Report Committee for Aman Singhal
Certifies that this is the approved version of the following report:

Place Me: Location-Based Mobile App for Android Platform

APPROVED BY
SUPERVISING COMMITTEE:

Supervisor: _____
Adnan Aziz

_____
Sarfraz Khurshid

Place Me: Location Based Mobile App for Android Platform

by

Aman Singhal, BSEE

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

December 2010

Acknowledgements

I would like to thank my professor, Dr. Adnan Aziz for his encouragement, support and technical guidance on this project. I would also like to thank Dr. Sarfraz Khurshid, for his input and advice as the reader for this report. Finally, I would like extend special thanks to my parents Shekhar and Renu, and sister, Anshu, for their unwavering confidence, encouragement and emotional support which was instrumental to the successful completion of my Master's coursework.

November 22, 2010

Abstract

Place Me: Location Based Mobile App for Android Platform

Aman Singhal, MSE

The University of Texas at Austin, 2010

Supervisor: Adnan Aziz

This report describes PlaceMe, a client side, mobile application built on the Android platform that provides personal location-based services such as location reminders, bookmarking, mapping and search nearby. The reminder system allows creating location based reminders, and alerts the user what he needs to do, when he is in the right place to do them. Bookmarking allows the user to virtually "save" places of interest while he is on the move and obtain driving directions. Mapping enables the user to visualize his relative geographic location in real time, and map the location reminders and bookmarks. Finally, search nearby exploits Google's powerful local search engine to allow finding and bookmarking nearby places such as gas stations, restaurants, etc, and retrieving map-based directions.

We first discuss the requirements and use-cases for PlaceMe, followed an introduction to the Android software stack. Next, we describe our design architecture,

implementation model, test strategy and key performance enhancements. Then, we evaluate and compare the performance of the Android platform across a set of standard micro-benchmarks. Finally, we conclude with a discussion of future development ideas and present our thoughts on prospects of app-based mobile computing.

.

Table of Contents

List of Tables

List of Figures

INTRODUCTION

The ubiquity of wireless networking and the trend toward component miniaturization have led to the evolution of cell phones from mere telephony devices to powerful mobile computing platforms that provide the basis for a host of other applications. Today's mobile phones are typically equipped with devices such as GPS sensors [1], Wi-Fi [2] and 3G wireless radios [3] capable streaming high bandwidth Internet content, touch-screen-based user interfaces [4], still and video cameras, Bluetooth transceivers [5], and accelerometers [6].

Similar to a computer, a mobile operating system provides the primary execution environment for applications on the phone. Analogous to programs on a PC, apps can be downloaded and installed on mobile phones. Because of the growing general purpose computing capabilities of mobile devices, combined with their increasing popularity and adoption rate, it is expected that hand-held mobile phones will become the next PC. These technology trends have enabled innovative, exciting and compelling mobile applications to become widely available, from gaming to multimedia to social networking. Hand-in-hand with the growth of the raw computing power of mobile phones, various middleware/OS platforms have evolved that allow developers to take advantage of the computing resources to create feature-rich applications that provide compelling user interfaces and functionality. A wide selection of proprietary and open-source mobile OS platforms exist, the most prominent ones being: Apple's iOS, Google's Android, Symbian from Symbian Foundation, RIM Blackberry OS, and Microsoft's Windows Mobile.

This report focuses on mobile app development for Google’s Android OS, a hugely popular open-source platform based on the Linux-kernel [7] and Java

development environment [8]. A growing number of cell phone manufacturers and vendors such as Motorola, Samsung and HTC, have adopted Android as the platform of choice for their products. In June 2010, Google announced that 68,000 apps are now available on the Android Market and the number hit the 100,000 mark before the end of July. According to the reported statistics, the number of apps downloaded to Android enabled mobile handsets reached 1 billion as of July 2010 [9].

Chapter 1: Mobility & Location Sensing

The ability to sense location is one aspect where a mobile device distinguishes itself from a traditional PC in terms of benefits and utility. Unlike a PC, a mobile device can be easily carried by the user and conveniently accessed for location based applications such as looking up driving directions, maps, etc.

The importance and usefulness of location-sensing has already been well recognized and accepted, with the popularity of GPS based navigation systems [1]. The form factor and portability of a mobile phone makes it ideal for such applications. Not surprisingly, most of the today's smart-phones have location sensing capabilities built in. Since people are most likely to use mobile phones when on-the-go and away from home or office, apps that leverage location-based services can add real value to the user and thus provide a good return-on-investment for the mobile device. A number of the most successful apps on the market today, use some element of location in a way that adds value to the user. Some popular examples are:

- Four Square, a mobile app that allows users to “check-in” at various consumer and retails outlets such as restaurants, malls, hotels, etc and earn reward points [10]
- Loopt, which allows users to track the location of their Facebook friends and receive alerts when they are nearby [11]
- Gowalla, which allows users to broadcast their location on Twitter, share photos of places they visit with their friends, and get localized coupons and offers [12].

Chapter 2: Place Me

This report presents Place, a mobile app targeted at Google's Android Platform. It provides a highly useful set of integrated location based services such as:

1. A reminder system that allows users to create location reminders and be alerted when they enter or exit the vicinity of a given location.
2. Ability to bookmark places of interest and quickly access driving directions to these places from their current location.
3. An interactive local map that allows users to map their current location in real time, and view bookmarked locations and reminders on the map.
4. An integrated search engine allows users to search for places, such as restaurants, bookstores, gas stations, etc., near their current location and get directions.

USE CASES & APPLICATIONS

This section discusses some interesting use-cases and applications that the Place Me app is envisioned for. We start by briefly describing the functionality provided by the different features of the app and then give user stories that highlight key requirements and design goals.

Location Reminders

Reminders are usually based on time. Frequently, however, time is not sufficient to capture the context in which the user wants to be reminded. Studies [13] have shown location to be useful element of context. A location aware reminder system can allow

users to set reminders based on where they are located, and whether they are entering or exiting a particular location. Users can "post" reminders to these places (similar in principle to virtual "post-it" notes). The Place-Its paper [13] presents a good survey of the usefulness of location based reminder system in mobile phones on a real, diversified sample of mobile phone users and describes some interesting usage scenarios. One of the interesting findings that came out of this study was that location can be used as convenient proxy for inferring other context that cannot otherwise be easily sensed. For example a person may have access to resources such as fax or copy machines at his workplace, but not at home.

The PlaceMe reminder system allows users to create reminders for specific places and receive alerts when they enter or exit a specified range of these reminder locations. Unlike the Place-Its reminder system, the location is specified as an address so user can set the reminder for new place without actually having visited there before. Here are a few everyday scenarios describing how PlaceMe location reminder system can be used.

User Story 1: Sally is a busy computer engineer who wants to be reminded to pick up groceries on her way back from work. She creates a location reminder for her workplace and sets it to trigger upon exiting from work. The list of the items she needs to buy is specified in the body of the reminder. As she is driving out of work in the evening, her phone rings and she is reminded to go to the grocery store before heading home, along with a list of items she needs to buy. Thus, the PlaceMe reminder system helps Sally stay on top of her tasks.

User Story 2: Bob is a sales executive who often travels internationally for business. Frequently, his trips include connecting flights at airports along the way. He uses PlaceMe app to plan his trip by setting up reminders for the terminals he needs to board the next flight from, at each airport along the way. Bob is scheduled to give a sales presentation at a customer site in London. He departs from Seattle on Monday morning, and has to change flights at airports in Dallas and New York on the way. When Bob arrives at Dallas airport his phone rings, and he is reminded to board the flight to New York from Terminal E. Similarly, upon reaching the airport in New York he is reminded to head over to Terminal C for the connecting flight to London. Finally, upon reaching London, he is reminded of the address to the customer site. The PlaceMe reminder system thus makes Bob's transit easier by providing vital information at the right time.

Bookmark Places

The bookmark feature allows users to save places of interest for convenient reference. The name and address of the specified place is saved in the list of bookmarked places. At any time the users can view a local map of the area and access turn-by-turn driving directions to bookmarked places from their current location and vice-versa. Consider the following scenarios.

User Story 3: Sam is manager of a local catering company which has a set of key customers around town who require catering services from time to time. He usually needs to make trips to customer sites when on call. Sam uses the PlaceMe app to bookmark addresses of his most frequent customers so he can quickly access driving directions while on the move. In addition, he also bookmarks addresses to major grocery stores in

the area where he shops for supplies. Bookmarking makes Sam's job much easier as he does not have to constantly keep track of the addresses and directions of different customer sites and grocery stores. It also helps saves valuable time needed to find addresses, and instead focusing on meeting last-minute customer requests.

User Story 4: Jack is planning a family trip to New York City for the weekend. He uses the Bookmark Places feature of the PlaceMe app to bookmark landmarks, tourist spots and attractions in his itinerary. On the day of his visit, he uses the built-in directions tool to obtain driving directions to the places he has bookmarked which allows him to navigate his way through the city efficiently. Upon exploring a particular attraction, when he needs to know how to get to the next spot from his current location, the convenient "get directions" feature allows him to view a local map of the area as well as obtain turn-by-turn directions to the next stop. Thus, the bookmark feature helps Jack make the most of his time.

In addition, Jack can also use the PlaceMe reminder system to further plan out his travel itinerary by creating location reminders for other nearby places of interest to explore upon arriving at a particular place, or be reminded of the next place to visit upon departing from the current spot.

Map Me

The Map Me feature shows the following items as icons on a local map of the area: (1) current location in real time (2) location reminders (3) bookmarked places. This allows the user to place the reminders and bookmarks in spatial or geographic context relative to their current location, providing a bird's eye view of their immediate

surroundings. Users can pan the map around and quickly zoom in and out to visually survey the area. In addition, the bookmark and reminder icons can be clicked to view more detailed information such as title, address, etc.

User Story 5: Continuing User Story 4 from above, suppose Jack is on his way to a tourist attraction and is passing through downtown Manhattan when he encounters a traffic jam. While waiting in traffic, he would like to view a map of the area to see if he can take any detours or short-cuts to avoid the downtown traffic. The Map Me feature allows Jack to view his current location on the map as well as his next bookmarked attraction, and enables him to quickly scan the area to find alternative routes which are less crowded. If he gets lost while taking the short-cut, he can always use to the "get directions" feature described in the Bookmark Places section above to find his way to the next attraction on his list from where he may be situated at the time.

Search Nearby

Search Nearby features a powerful integrated search engine that allows users to find specific places such restaurants, bookstores, coffee-shops, gas-stations, etc, near their current geographic location using keywords. The results are displayed as a list, showing the name, street address, phone number and approximate distance in miles from the current location of each place found. User can also obtain driving directions to any place found or bookmark the place for future reference.

Note that the functionality provided by this feature can also be accessed by performing a Google search using a web-browser app on the mobile phone. However this requires considerably more typing, clicking, zooming, and waiting for the web page to

load, making it very time-consuming and cumbersome to find things quickly. The search nearby feature provides an integrated mobile solution that makes it faster and easier to locate nearby places and find relevant information quickly.

User Story 6: Continuing User Story 5 from above: Jack and his family are on the road visiting places on their itinerary when they decide to have pizza for lunch. Jack uses the search nearby feature to quickly locate pizzerias nearby and retrieve detailed driving directions to them. Thus, Jack and his family are able to stop over at a nearby pizzeria just in time for lunch.

User Story 7: Continuing User Story 3 from above: Sam is driving to a customer site for catering. While on his way, he realizes he is short on portable burners. He uses the search nearby feature to locate a nearby Wal-Mart and makes a quick stop to purchase the additional burners. Thus, Sam is able to meet his customer needs in a timely manner.

User Story 8: Dorothy is visiting Austin over the weekend to see her son who lives in the dorms. She decides to spend the night in town to attend a football game the following day, and needs to find a nearby motel which has a vacancy. Dorothy uses the search nearby feature in PlaceMe, to find nearby motels, and call the provided phone number to check availability and make a reservation for the night.

The user stories presented above capture some of the different ways the Place-Me app can make the people's lives easier. There are many more applications scenarios where one might find the services provided by the app useful.

GOALS & REQUIREMENTS

Based on the use cases discussed above, in this section, we identify and discuss some of the high level design requirements with regard to the nature of services provided by the app and describe how we strive to meet them in the design of Place Me. The rationale behind some of the key features is also presented. Oinas-Kukkonen's paper on mobile app design outlines 7 principles of highly goal driven mobile apps [14]. We use these principles to identify, organize and present the design goals and requirements below.

Mobility & Usefulness

The purpose of the Place Me app is to provide useful location services when the user is on the move. The key services provided by Place Me described in the previous section, namely, Locations Reminders, Map Me, Search Nearby, and Bookmark Places, are designed around this principle and fundamentally location oriented. The user stories discussed in the previous section illustrate the usefulness of the different features of PlaceMe for a mobile user.

Relevance

Relevance is the appropriateness of information presented to the user based on his current context and needs. The information and services provided must to be real-time and location specific to meet the needs of the mobile user. Here are some of the ways in which the relevance goal is achieved in the different services provided by the app.

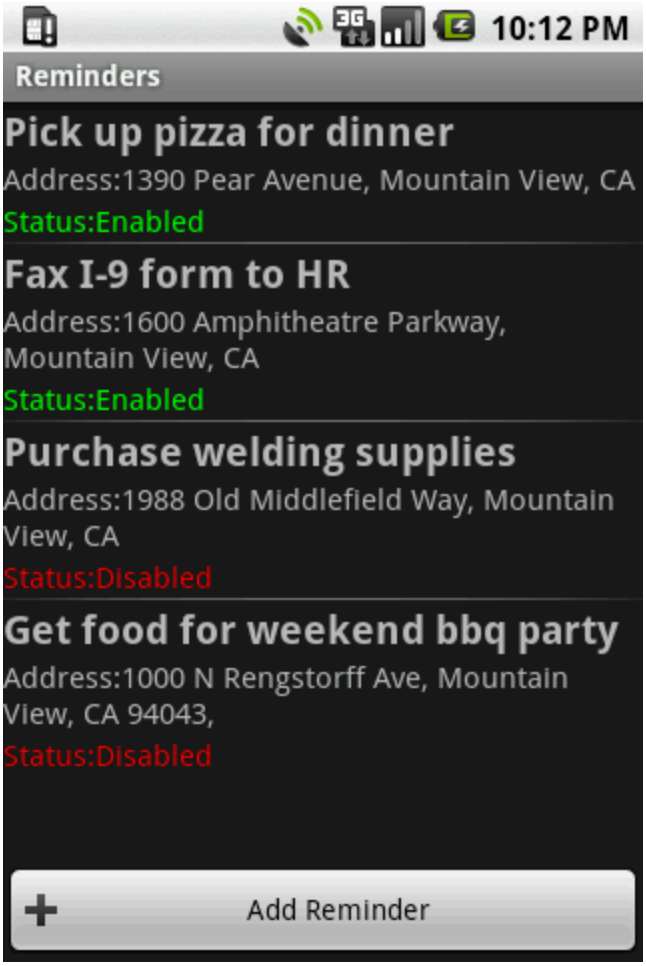

Figure 1: Reminder List

The app monitors the user's location in real time and triggers only those reminders which are set for the user's current location context, while suppressing the other reminders. In addition, the reminder listing, shown in the figure above, only displays the most relevant elements of the reminders created by the user, namely, the title, location and status, while hiding the other attributes such as the trigger range or the body of the reminder. The title and location identify the "what" and the "where", while the status indicates whether the reminder is enabled or disabled. The enabled/active

reminders (green) appear at the top of the list to enhance visibility of items which the user cares about the most, while the disabled ones (red) show at the bottom.

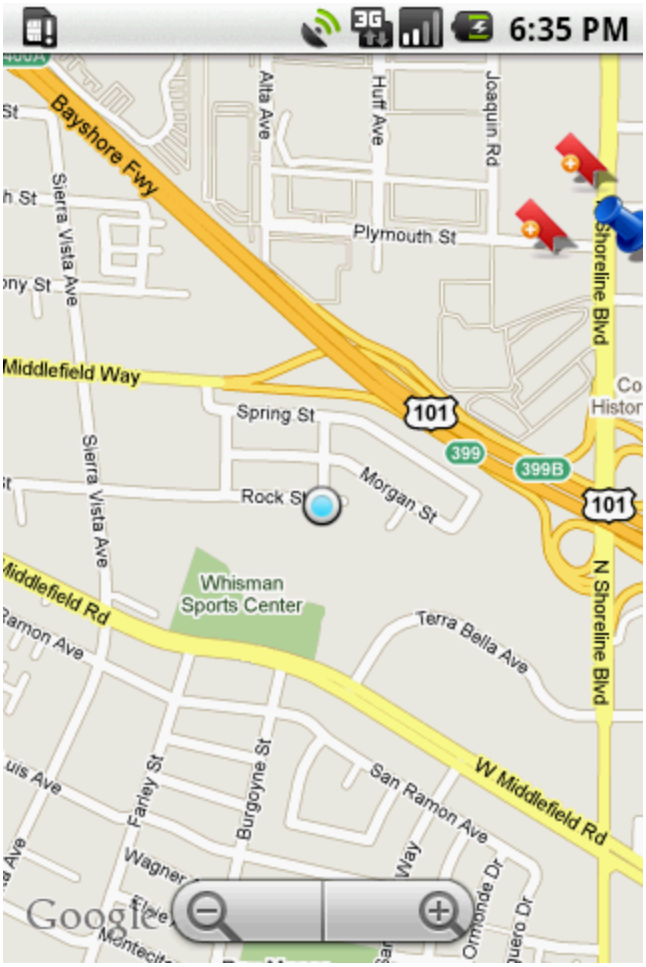

Figure 2: Map Me

The Map Me service shows a map of only the area immediately surrounding the place where the user is a currently located. As shown in the above figure, the user's location is indicated by a blue dot in the middle of the screen, and the map auto-scrolls and refocuses to reflect the user's current location in real-time if the user is mobile when

viewing the map. This provides an up-to-date picture of the user's current location context at all times.

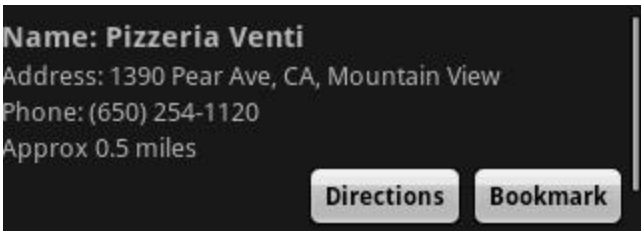

Figure 3: Search Nearby Result

Search Nearby conducts a key-word based search for places in the user's immediate vicinity as these are places he will he be most interested in. The search results display the most relevant attributes of the places found, namely, the name, street address, phone number and approximate distance to the place from the user's current location. The name and street address are useful for physically locating the place as the user is driving, while the phone number is useful if the user needs to contact the place for finding more information such as checking the availability of rooms in nearby motels.

In addition, the Bookmark Places and Search Nearby services allow users to obtain directions to or from the location where the user is currently situated to the specified place.

Ease of Use & Fluency of Navigation

PlaceMe is designed to be used when the user is mobile. A simple, intuitive user interface that minimizes the need for manual data entry is therefore essential. Moreover, the services and features should be readily accessible and the user should not need to

spend time reading manuals to figure out how to use the basic functionality. This philosophy of simplicity, accessibility and user-friendliness is embodied in the design of the app.

Figure 4: Home Screen

The primary services provided by PlaceMe are accessible from the home page of the app, which makes it quick and easy to navigate to across tasks as shown in the figure above.

Figure 5: Add Reminder & Add Bookmark Buttons

The reminder and bookmark lists, each have an “Add Reminder” and “Add Bookmark” button respectively, conveniently placed at the bottom of the screen which makes it easy to add new reminders or bookmarks. In order to view more details or change the attributes, users can simply click the appropriate reminder or bookmark from the list. This helps reduce the number of buttons on the screen keeping the interface simple.

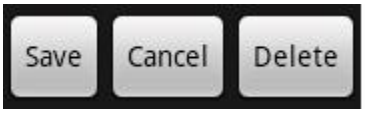

Figure 6: Save, Cancel & Delete Buttons

The bookmark and reminder update screens have options to save, cancel (ignore any changes) or delete the corresponding bookmark or reminder. This provides a single, unified interface to perform edit, update or delete operations making it simple and easy to make changes. Alternatively, quick edit or deletes can be performed via a context menu as shown in the figure below.

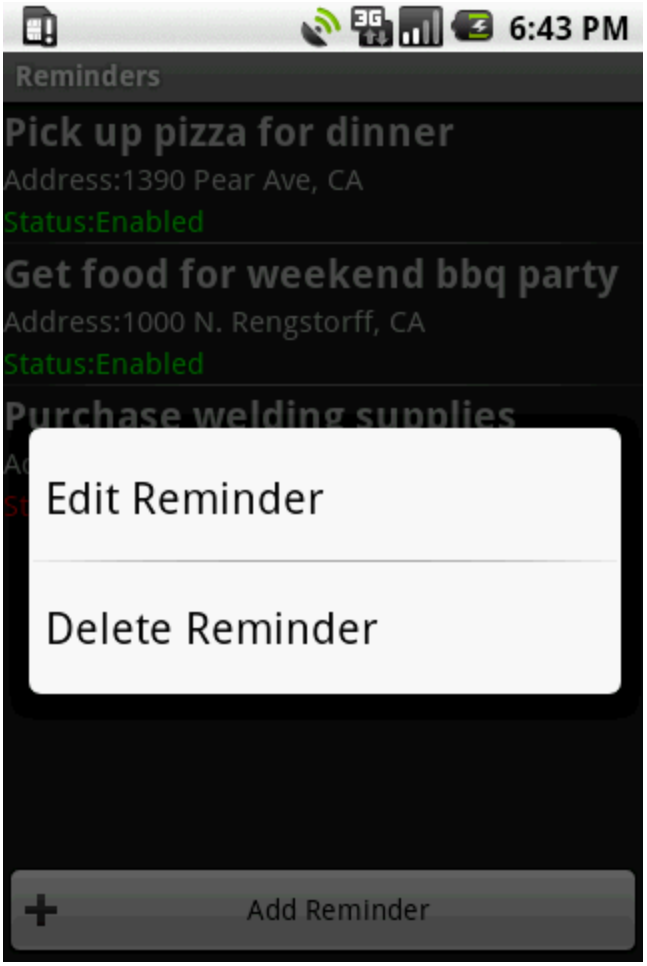

Figure 7: Context Menu

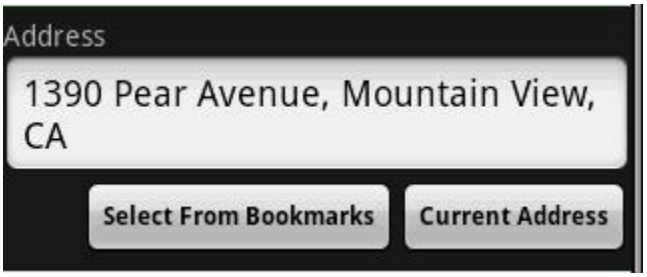

Figure 8: Address Entry Options

If the user is already at the location for which the reminder is to be created for, the "use current address" button can be used to automatically fill out the address field of the reminder for the user's present GPS location. If the reminder is to be set for a place that is already saved as a bookmark, the "select from bookmarks" option can be used to allow the user to pick the address from a list of bookmarked locations as shown in the above figure. The "use current address" and "select from bookmarks" are convenience features which make it easy to specify the address without requiring manually typing out the address text every time. This functionality is implemented using Google's geo-coding APIs [15] discussed in the Design and Implementation section.

Figure 9: Reminder Notification

Reminder notifications are displayed in the notification bar at the top of the screen for greater visibility. As shown in the figure above, the title of the reminder flashes in the notification bar along with any associated alerts such as a ringtone. The notification bar can be expanded to directly navigate to the reminder screen where the user can view and update the details of the reminder such as the list of items to buy from the grocery store.

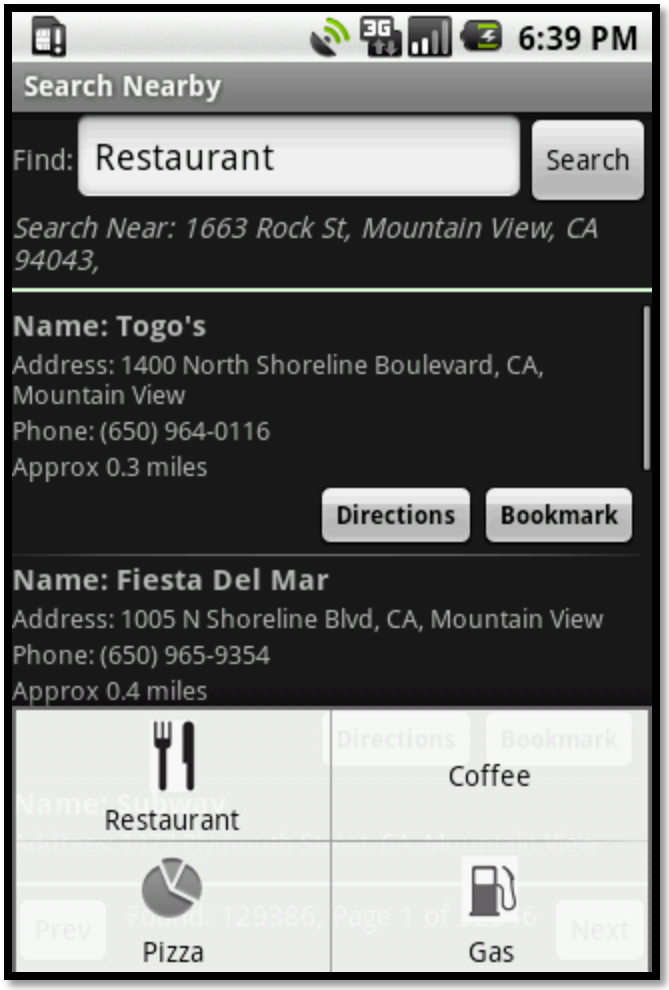

Figure 10: Search nearby screen

On the search nearby screen, users can enter their search query in a text box to initiate a local search. Alternatively, they can select a general search category from a pop-up menu which minimizes typing making it convenient to search quickly. Additionally, each place found has a “bookmark” button so the user can quickly save it for future reference and a “directions” button for obtaining map-based or turn-by-turn driving directions.

User-Centeredness & Personalization

Users' needs vary based on their situational or locational context. Moreover, no two users are the same and can have different preferences and priorities. Some level of flexibility must therefore be provided to allow users to configure and customize the app according to their individual needs. Here we discuss some of the user-centric personalization features of PlaceMe.

When creating reminders, for example, users can specify:

- A custom title that serves as a mnemonic to help them quickly recall what the reminder is for, which is also displayed in the reminder list. For example, "pick-up groceries from HEB", "call home", etc.
- The type of notification alert they would like to receive, which can be any combination of ring, vibrate or LED flash alerts. For example, a user may prefer to use vibrate or flash notifications when he is at work, while a ring would be best when the user is in a more noisy environment such as driving in his car.
- The radius (in miles) around the specified location and the trigger event (entry or exit). A small radius would cause the exit reminder to trigger quickly as soon as the user leaves the place, whereas, an entry reminder would trigger only when the user approaches very close to the place. For example, one user might like to be reminded to pick-up a package from the apartment office as he is entering the apartment complex after work, while another might like to be reminded to do so as soon as he leaves work so he can plan to make it home in time to before the apartment office closes.

Figure 11: Reminder service start/stop menu

Reminders can be enabled or disabled by the user. On occasions, however, the user might choose to turn off the reminder service altogether, for example, when one is on vacation. Instead of forcing the user to disable each reminder in turn and remembering to enable the relevant ones, the app provides a convenient way to turn off the reminder service with a single button shown in the figure above. Once turned off, the user will not be reminded until the service is turned on again.

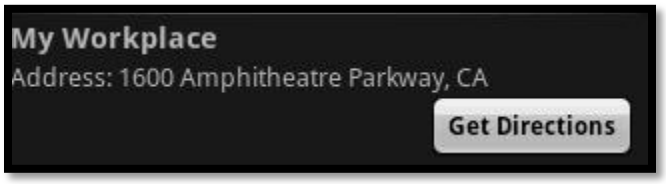

Figure 12: A bookmarked location

The bookmark service allows users to specify a title for the bookmarked location such as for example, “My Home”, “My Workplace”, “Grocery store”, etc. to further personalize the saved locations by labeling them based on his needs for easy reference. The directions feature provides a convenient way for the user to find their way back to bookmarked place without having to remember its address. For example, on his way back

from the airport Sam wants to stop over at his favorite restaurant for dinner, but may not remember how to get from the airport to the restaurant. Since he has his favorite restaurant bookmarked, Sam can get directions to it from wherever he might be. In addition, while viewing directions for bookmarked places or places found using search nearby service, users can choose between map-based or text-based turn-by-turn directions. Moreover, map-based directions can be viewed in regular or satellite views based on the user's preference.

When viewing maps, in addition to panning around, users can zoom in and out. The zoom level, once set by the user, is remembered. The next time the map is viewed or updated to reflect the user's current location this customized zoom setting is restored and used. This saves the trouble of constantly having to re-adjust the zoom level, and can be quite handy especially if the user needs to refer to the map while driving.

DESIGN & IMPLEMENTATION

Before we delve into the application design of PlaceMe, let us first understand the infrastructure, APIs and development tools provided by the Android platform that can be used to implement an app that meets the design goals.

Chapter 3: Android Platform Architecture

Android is an open-source software platform developed by Google, for mobile app development on devices powered by the Android OS. It is a complete software stack that provides all the middleware needed to run end-user applications on mobile devices such as: device drivers, OS, core libraries, an optimized virtual machine, Java Native Interface (JNI), and a complete Java development environment. This section provides a detailed introduction to the Android framework and describes the platform architecture, execution model, and key concepts pertinent to the design of the Place Me app, which are more generally applicable to other apps as well. As shown in the figure below, the Android software stack is a tiered architecture that consists of 5 principle layers [16].

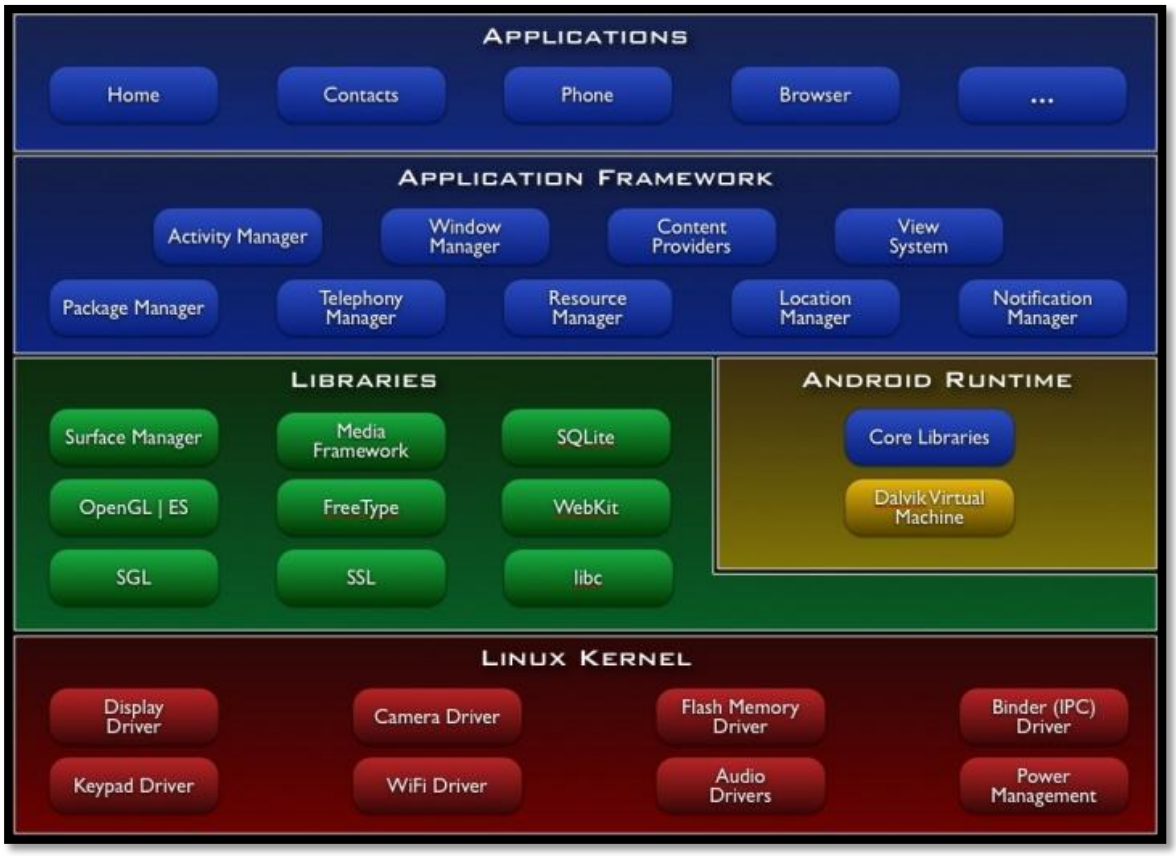

Figure 13: Android Architecture

KERNEL

Android is built upon Linux 2.6 kernel which serves as the hardware abstraction layer. Linux is used since it provides proven and robust, low-level system infrastructure components such as memory and process management, security, network stack and hardware driver model. Original Equipment Manufacturers (OEMs) can thus bring-up Linux on their system and have the drivers running before loading the other components of the stack [16].

LIBRARIES

On top of the kernel layer are the native libraries written in C and C++, which provide most of the real power of the Android platform. The surface manager is responsible for composing, coordinating and rendering surfaces on the screen from windows owned by different applications, running in different processes in tandem, and ensuring the pixels show up correctly on the screen. OpenGL/ES and SGL are the core 3-D and 2-D graphics libraries respectively. The 3-D graphics can be accelerated in hardware if a 3-D chip is present. Most of the applications commonly use 2-D graphics; however, the platform allows combining 2-D and 3-D graphics as well. The media framework provides all of the audio and video codecs responsible for delivering a rich media experience. FreeType is used for managing and rendering fonts on the screen. The open-source SQLite relational database management system is used for most of the core data storage on Android. It allows creating in-memory databases for very efficient data storage and retrieval. WebKit is an open-source browser engine (also used in Google's Chrome and Apple's Safari browsers) for optimized rendering on small screens in mobile devices [17].

ANDROID RUNTIME

On the same level as the libraries discussed above is the Android runtime, which is designed for running Java programs in resource constrained, embedded environments with limited computational power, battery life, and memory. One of the main components of the Android Runtime is the Dalvik virtual machine. The Dalvik VM is an optimized byte-code interpreter for efficient byte code execution on small-scale processors used in mobile devices. The Java class and JAR files are translated into ".dex"

files at build time, for execution on the Dalvik VM. The efficiency of Dalvik makes it possible to run each application as a separate process in its own instance of the Dalvik VM. Among other benefits, this simplifies memory management and improves platform stability. For example, if one application crashes it does not affect other applications as they are running in separate processes each in its own instance of the Dalvik VM. The Core Libraries component provides commonly used collection classes, input-output libraries, utilities, and tools written in Java [17], [18].

APPLICATION FRAMEWORK

This layer consists of a set of tools and APIs written in the Java programming language which are used by the application developers. Here we discuss some of the main components of the application framework. The activity manager is responsible for application life cycle management and maintains a common back-stack for providing smooth navigation between apps running in different processes. The package manager tracks and manages the location and capabilities of applications installed on the phone, including those provided with the phone as well as those downloaded by the user. The window manager is a Java implementation that abstracts lower level services provided by the surface manager for managing windows. The telephony manager provides the core Java APIs used for implementing the phone service. Content providers are a common interface for sharing data between different applications. For example, the contacts data stored in the Contacts application can be accessed by any application that needs to use it by means of content providers. The resource manager stores non-code related artifacts of the application such as localized strings, bitmaps, or external resources such as images, video or audio files. Finally, the view system implements the building blocks of user-

interfaces, provides APIs for drawing layouts and UI elements, and handles event-dispatching [17].

APPLICATIONS

The top-most layer in the stack is the Applications layer. All applications are written in Java and used the same set of APIs provided by the Application Framework. This included applications that are shipped with the phone such as Home, Dialer, Contacts, Browser, etc. as well as those developed by the programmers [17].

Chapter 4: Anatomy of an Android App

This chapter briefly discusses the basic components of an Android app and defines the key concepts and vocabulary needed to understand the design of PlaceMe app.

VIEW

Views are the fundamental building blocks for creating user-interfaces. A View typically consists of the content visible to the user on the screen such as a button, text field, etc. It is the point of user interaction and handles UI events such as a button press. Views are grouped into a hierarchical structure to form different layout schemes such as a lists, tables, etc which organize the Views into specific pattern for rendering [17]. Layouts and Views are typically specified in XML files.

ACTIVITY

An Activity is essentially a piece of user-interface that consists of a set of related tasks a user can do in one screen. For example, a mail app can be divided into 3 basic activities: (1) Mail list Activity that shows all received mails (2) Mail view Activity that shows a single mail message (3) Compose Activity that allows creating and sending outgoing messages [17].

INTENTS

Intents are the fundamental message passing constructs in Android which allow communication of data and action between and among different system components such

as: Applications, Activities, Services, etc. For example, when a new email is received, Intents are fired from the mail listener service to update the mail list screen to show the newly received messages.

Apps can also register to receive specific kinds of Intents (generated internally or externally) in order to wake up and execute code when the appropriate Intent is received. For example, say the user wants to pick a photo to attach to a mail message. An Intent is fired that looks for the best available source of photos. It may determine that the photo gallery is the best match and use it. If later, a better source of photos is added, such as an online web album on Flickr or Picasa, the photo gallery is replaced by the web album as the preferred source. This late binding between action and action handler allows components to be re-used or replaced at run-time. Any task triggered by an Intent is therefore an opportunity to replace or re-use a component [17]. For instance, an Intent for viewing emails can first be sent to the mail service which fetches any new mails from the server and updates the mail listing before opening the inbox.

SERVICES

Services are background processes launched from Activities that typically perform long-running tasks and have no user interface. For example, a music player can be started as a service and keep playing music while the user may be checking emails. Other Activities or Applications can also bind to the service for performing specific tasks such as pausing, rewinding or fast-forwarding the music [17].

Chapter 5: Place Me App Architecture

In this section we discuss the design and architecture of the PlaceMe app. The figure below shows a static UML class diagram of the PlaceMe app, illustrating the key Activities, Services, Intents and inter-component dependencies (only a few class methods/attributes have been shown in the figure below to reduce clutter and illustrate the high level class structure in a clean and concise manner).

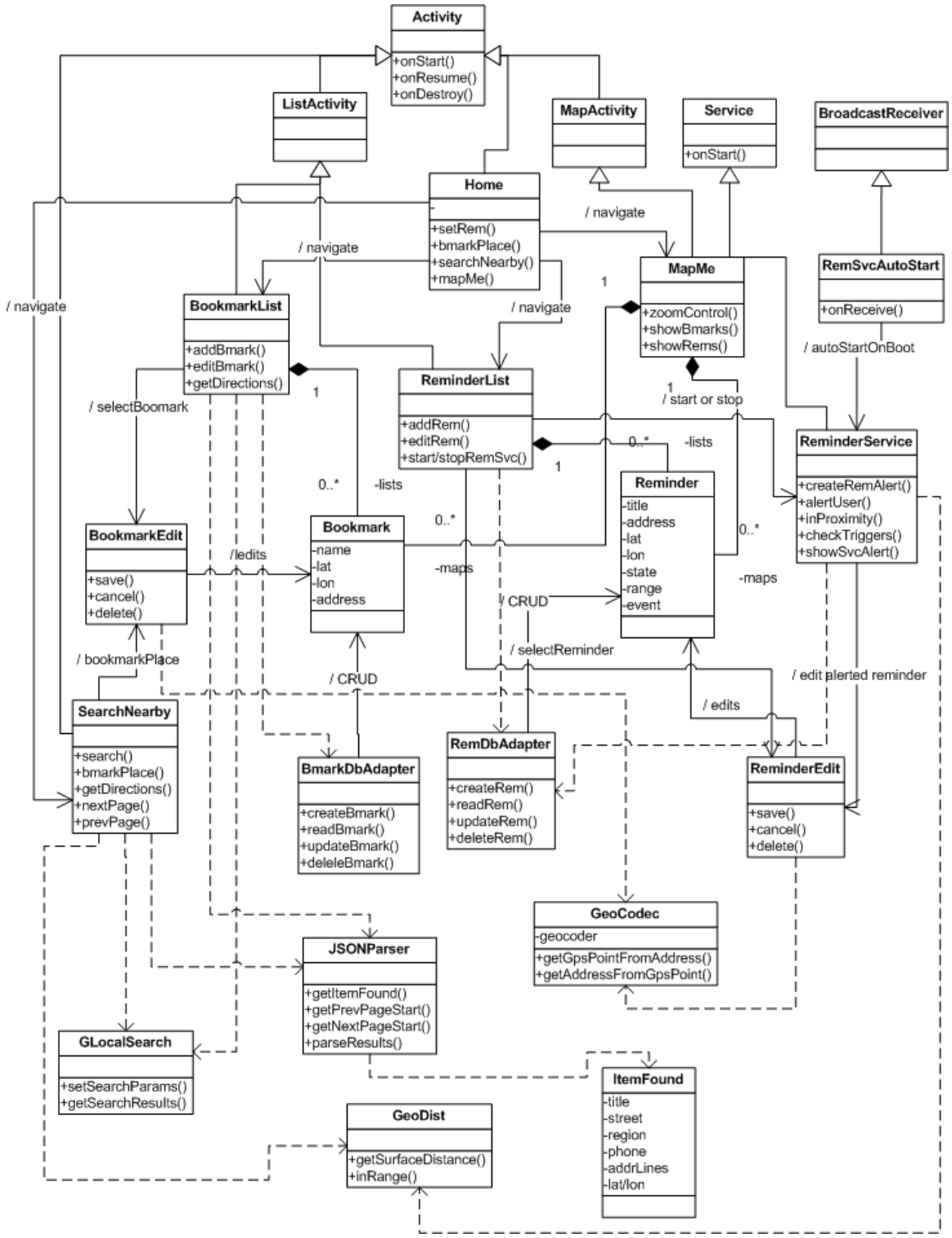

Figure 15: PlaceMe UML class diagram

REMINDER SYSTEM

The reminder system consists of 3 key components: (1) a reminder database that stores information about each reminder created by the user, (2) a background service that triggers reminders based on the user's location, and (3) a notification system that alerts the user when the reminder is triggered. Each component is described in more detail below.

Reminder database:

The user-created reminders are stored in a Java-based SQL database which contains the following information for each reminder entry (the SQL data-types for each field are indicated in parenthesis below):

1. Title and body of the reminder (text)
2. Address the reminder is set for (text)
3. GPS latitude and longitude coordinates for the address (real)
4. A radius (in miles) around the GPS coordinate that defines the reminder zone (real)
5. Trigger events for the reminder: entry and/or exit from reminder zone (integer)
6. The current state of the reminder: enabled or disabled (integer)
7. Type of alert notification for reminder: ring, vibrate or LED flash (integer)

Reading and writing to the database is performed using the ReminderDbAdapter driver class which implements the SQL statements for performing database operations. The figure below shows the schema for the reminders table.

rowId | title | body | latitude | longitude | address | state | alert type | radius | event

Figure 14: Reminder database schema

The latitude and longitude coordinates are typically measured and expressed in degrees, minutes (1 minute = 1/60th of a degree), and seconds (1 second = 1/3600th of a degree). For programmatic convenience we convert and store the latitude and longitudes in the form: [+-] DDD.DDDDD where D is degrees. The sign indicates western or eastern hemisphere (for longitude), and northern or southern hemisphere (for latitude). Moreover, Android's Location library provides built-in support for this notation [15].

Reminder Service:

This is a background service runs in a separate process and can be started or stopped by the user. It uses a tracking algorithm that iterates through all the enabled reminder entries in the reminder database and triggers notifications when user enters and/or exits any reminder zones based on the specified trigger event. It also creates reminder notifications based on the user specified alert type.

Here we discuss the tracking algorithm that determines when reminders should be triggered as it is central to the correct operation of the reminder service. The algorithm uses the following inputs: (1) GPS coordinate the reminder is set for and the radius of the reminder zone, both of which are obtained from the reminders database (2) GPS coordinate of the user's current location (3) GPS coordinate of the user's last location. The user's current location is obtained from the Android's Location Manager Class and

the last location is value of the previous “current” location when the user’s location changes. Entry and exit events can then be defined in terms of these inputs as follows:

- Entry event: An entry event occurs when the user’s last location was outside the reminder zone, and the current location falls inside it.
- Exit event: An exit event occurs when the user’s last location was inside the reminder zone, and the current location falls outside it.

The flowchart below depicts the conditional logic used by the reminder service to trigger reminder alert notifications.

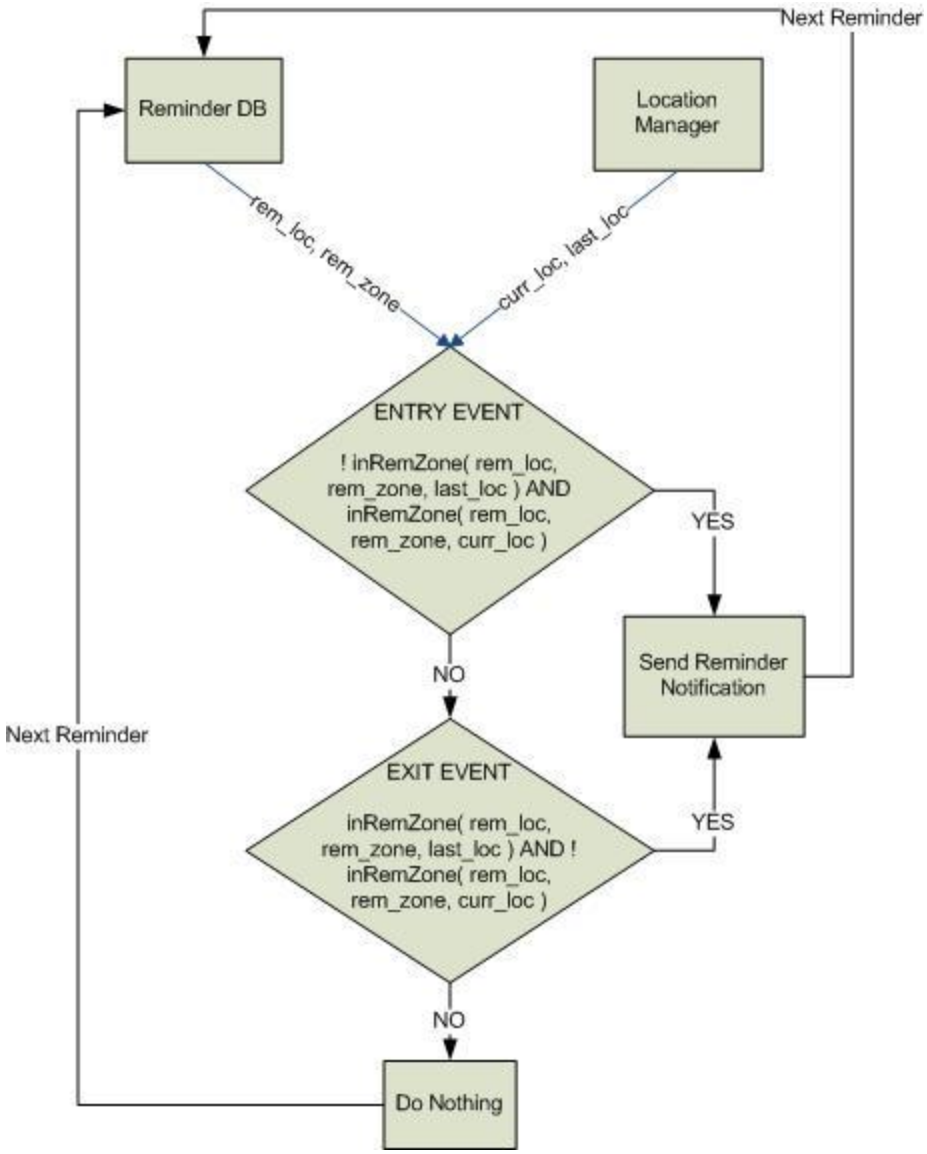

Figure 16: Tracking algorithm

The reminder service allows multiple reminder alerts to trigger simultaneously if two or more reminder zones overlap or are directly adjacent.

Reminder Notifications:

As mentioned earlier, services in Android do not have a user-interface component. The purpose of the reminder notification system is to make the actions performed by the reminder service visible to the user such as when reminder service is started or stopped or when alerts are triggered. The reminder alerts created by the reminder service are based on the user-specified alert type, namely, ring, vibrate, flash or any combination of the above. The notification system is responsible for updating the user interface when these alerts are triggered (ex: showing the reminder), executing the appropriate notification action (ex: ringing the phone), and handling user interaction in response to triggered reminder alerts (ex: stop notifying when the user acknowledges the alert).

When the reminder service is started, the notifications bar flashes a ticker text indicating the service has been activated and displays a persistent icon while it's running. Similarly, when the service is stopped, the ticker text is updated and the icon is cleared. This provides visual feedback to the user regarding the state of the service as shown in the figure below.

Figure 17: Reminder service start/stop notifications

In addition, expanding the notification bar and clicking the icon navigates to the reminder listing screen. The behavior is similar to how other services in Android behave

such as the network service which shows a bar icon indicating 3G/wireless network connectivity status shown in the figure below.

Figure 18: Reminder alerts

When reminder alerts are triggered, a ticker text flashes showing the title of the reminder and a bulb icon appears for each triggered reminder. At the same time, any sensory alerts (ring, vibrate, LED flash) associated with the reminder are started and remain active until the user responds by clicking the notification bar indicating that he has received the alert.

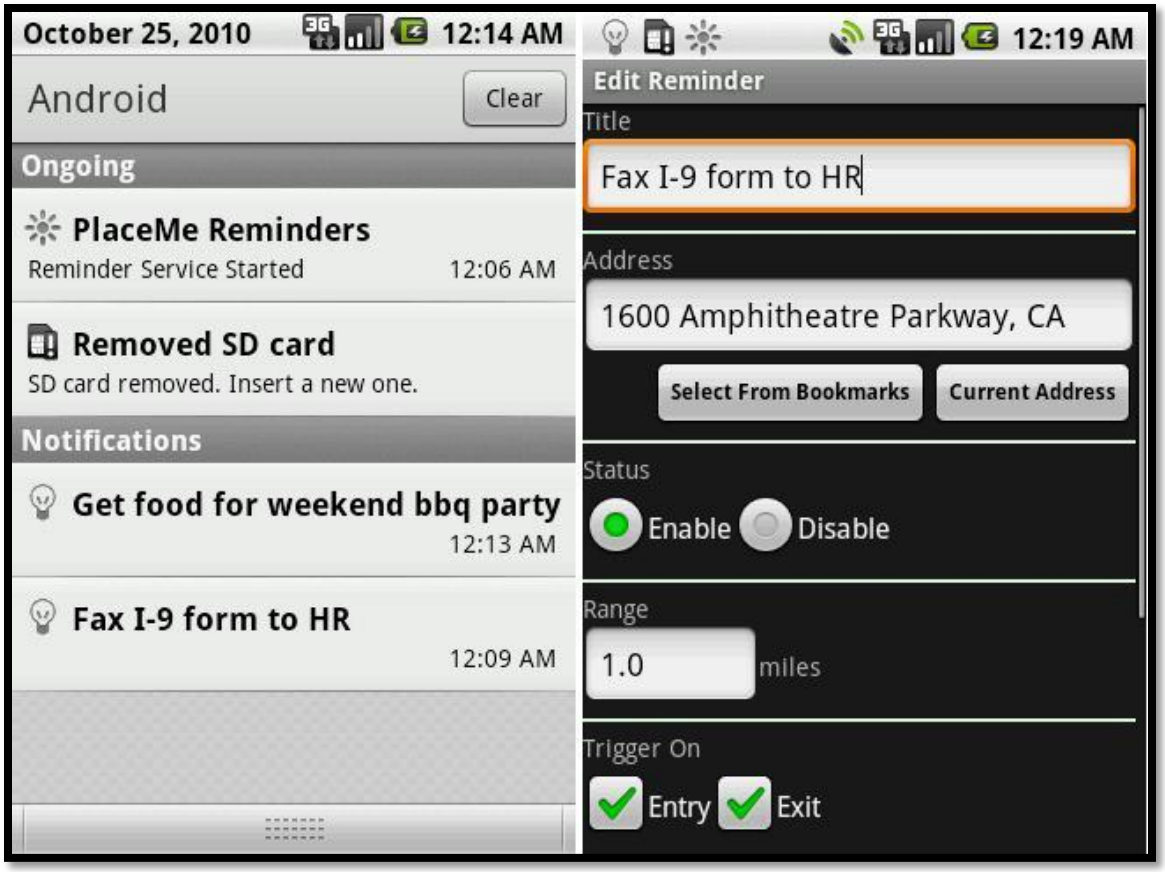

Figure 19: Expanded reminder alerts

The notification bar can be expanded to see the titles of triggered reminders and the time when they were received, while clicking on them fires an Intent that navigates to the corresponding Reminder Edit Activity where the reminder can be viewed, edited or disabled. The figure above depicts the expanded notification bar showing the triggered reminders (left) and the result of clicking on the “Fax I-9 form to HR” reminder entry (right) which brings up the associated reminder edit screen.

SEARCH NEARBY SERVICE

The Search Nearby feature allows searching for local business around the user's current location, bookmarking them, as well as obtaining directions to/from these places. As mentioned earlier, this feature leverages the local.google.com web-based search service and consists of the following key components:

(1) GLocalSearch class which creates and issues the Google search queries

(2) JSONParser and ItemFound classes, for parsing the results of the Ajax query, and populating the search results as a series of virtual pages.

(3) The Search Nearby Activity, which uses the above classes to implement the local search feature in PlaceMe.

These classes use the Google Ajax APIs for providing the search functionality and Google Maps for displaying directions.

Google AJAX Search API

Google exposes a RESTful interface to their Ajax Search APIs for application developers working in non-JavaScript environments. We have used this API interface for implementing the local search feature in PlaceMe. The search server endpoint is accessed by a base URL containing standard and searcher specific Common Gateway Interface (CGI) arguments which define the search query and scope. The HTTP GET protocol is used to obtain the search results in JSON format with embedded HTTP response result codes [19].

The standard base URL for accessing Google Local search service is: http://ajax.googleapis.com/ajax/services/search/local. The specific CGI arguments [19] we used to construct queries in Search Nearby are described below:

- q : Specifies the search term, for example: q=gas%20station , where %20 is the ASCII value for “space”
- v : Specifies the search protocol version number, which is currently 1.0
- rsz? : An optional argument that specifies the number of results to be returned. In Search Nearby we set rsz=8. Eight is the maximum number of search results that can be obtained in a single Ajax query.
- start: Specifies the page start index of the first search result in the JSON cursor object. For example, start=8.
- sll? : An optional Local search specific argument that specifies the center point of search as latitude, longitude coordinate pair, ex: 37.412006, -122.084095. In case of Search Nearby, search center point is always the user’s current location.

Thus, an example Ajax search URL for finding “pizza” places near a GPS location defined by 37.412006, -122.084095, would look as follows:

http://ajax.googleapis.com/ajax/services/search/local?v=1.0&q=”pizza”&sll=37.412006,-122.084095

The GLocalSearch class uses the user’s current location and the specified search term to build a Local search URL with the arguments defined above, opens a Java network socket and issues the query. The results of a local search query are returned as a simple JSON object shown in figure below [19]:

```
{
  "responseData" : {
    "results" : [],
    "cursor" : {}
  },
  "responseDetails" : null | string-on-error,
  "responseStatus" : 200 | error-code
}
```

Figure 20: JSON result format

The cursor in the object above consists of an array containing the page start indexes, total number of results, and the page index of the current page, as shown in the example below [19].

```
"cursor": {
  "pages": [
    { "start": "0", "label": 1 },
    { "start": "4", "label": 2 },
    { "start": "8", "label": 3 },
    { "start": "12","label": 4 } ],
  "estimatedResultCount": "48758",
  "currentPageIndex": 0,
  "moreResultsUrl": "http://www.google.com/search..."
}
```

Figure 21: JSON Cursor Object

The JSONParser class uses Android's JSON library [15] to parse the results into an Array of ItemFound objects. Each individual place found is encapsulated as an ItemFound object. The search results are then rendered as a set of virtual pages. The figure below shows the result of performing a search for "google" near 1663 Rock St., Mountain View, CA 9403 (user's current location) shown in the "Search Near: " text below the search box. The total result count and page number are displayed between the page navigation buttons at the bottom of the screen.

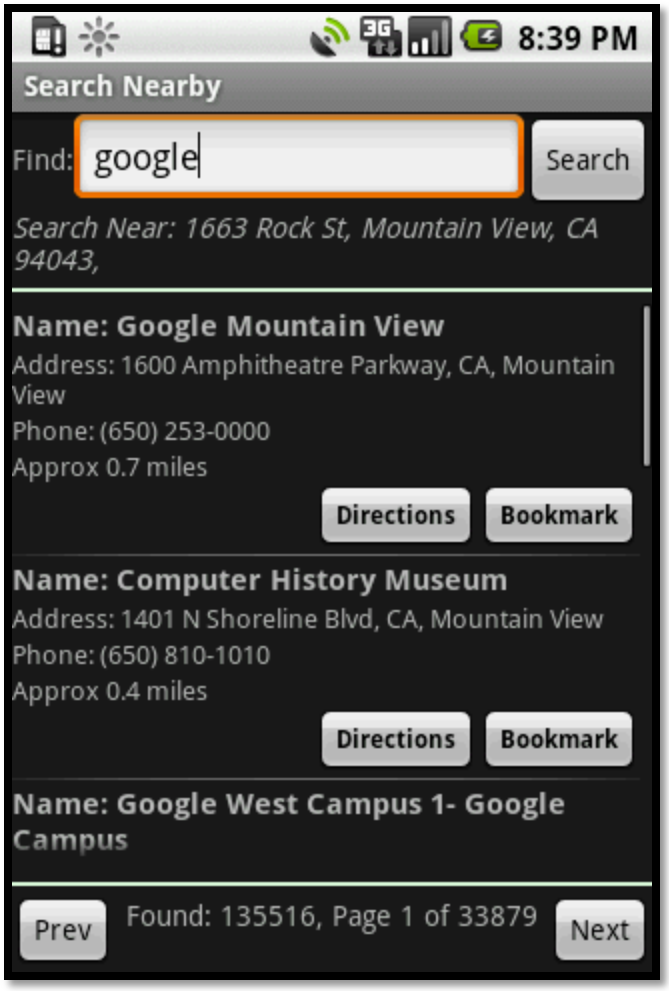

Figure 22: Search nearby results

The directions feature uses "mobile" Google Maps website (Google Maps optimized for handled devices) for showing directions. The URL to Google Maps website for showing the directions to and from a place is fetched from the JSON results returned by the Ajax query. When the directions button is clicked an Intent is fired to open the built-in Android browser app and navigate to the directions URL which displays the map-

based directions showing the route. There is also an option to access text-based turn-by-turn directions.

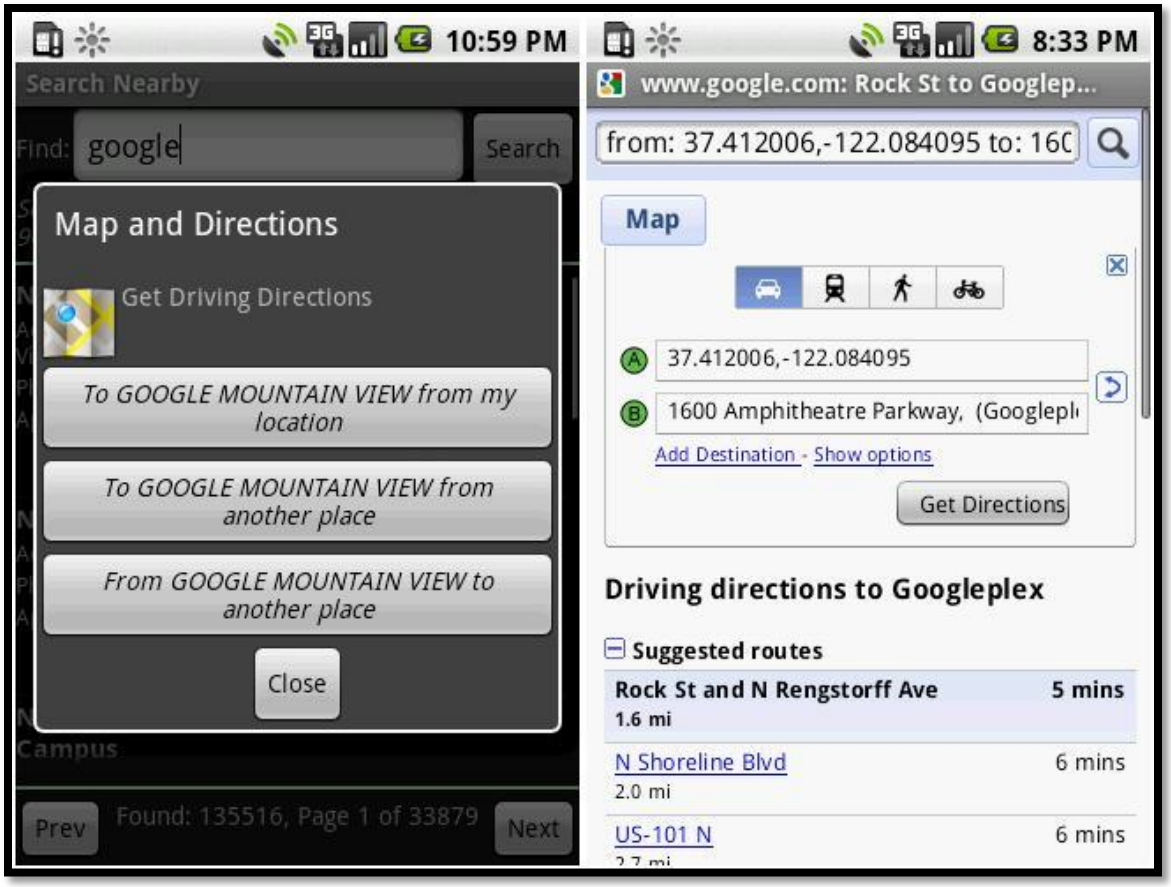

Figure 23: Directions feature in Search Nearby

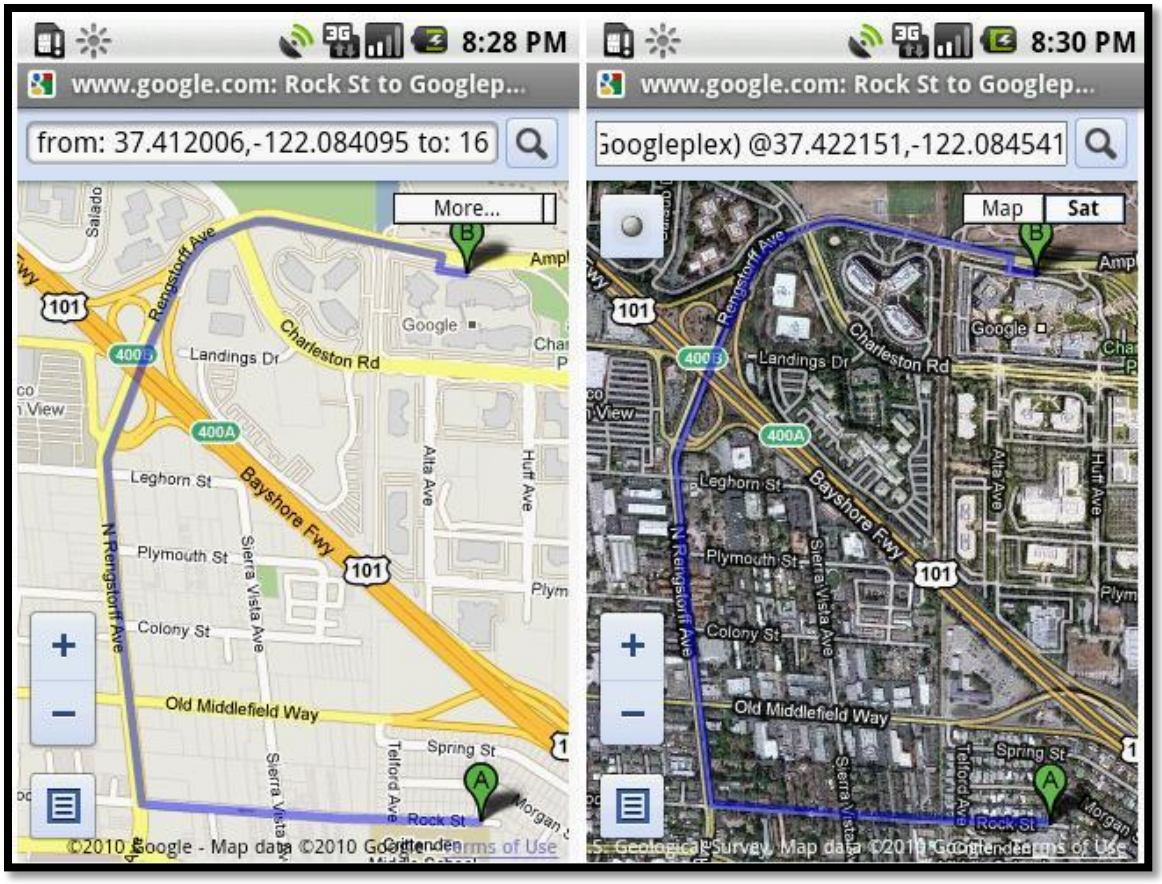

Figure 24: Map-based directions

The figures above show the result of accessing the directions to “Googleplex”, the first search result in Figure 23 above. One of the benefits of leveraging the Google maps website is the that the integrated Google map features such as satellite view, text based directions (driving, walking, biking, etc), can be easily accessed without implementing those separately in the PlaceMe application code. This allows greater flexibility and decoupling, for example, if Google makes improvements to their map website in the future, those would be directly available to users of PlaceMe without requiring any application changes.

BOOKMARKING LOCATIONS

The bookmark service allows saving locations of interest and retrieving directions to them. Any user specified address can be bookmarked including the user's current location. In addition, the bookmark service is fully integrated with the reminder system and the search nearby service, which makes it possible to: (1) fill the address field of the reminder from the list of bookmarked locations, and (2) directly bookmark places shown in search nearby list. Similar to the reminder system, an SQL database is used to store all the bookmarked locations.

Bookmarks database

Bookmarked locations are stored in a Java-based SQL database which contains the following information for each bookmark entry. The SQL data-types for each field are indicated in parenthesis below:

1. User specified name of the bookmarked location (text)
2. Address the bookmarked location (text)
3. GPS latitude and longitude coordinates for the address (real)

Reading and writing to the database is performed using the BookmarkDbAdapter Class, which implements the SQL statements for performing database operations. The figure below shows the schema for the bookmarks database table.

rowId | name | latitude | longitude | address

Figure 25: Bookmark database schema

Directions:

The JSON results returned by the Google Ajax search query contain the URL of the Google Map directions to/from a place, as discussed in context of the Search Nearby feature where we retrieved directions to nearby places. In this case, however, we want to obtain directions to a single bookmarked place that we know the address of. In order for this to work, a single result must be returned when the search is performed. To accomplish, this we build our Ajax search string using the following CGI parameters. For directions to a bookmarked place from current location: (1) search term: text address of bookmarked place (2) search center: GPS coordinate of current location. For directions from a bookmarked place to current location: (1) search term: GPS coordinate of current location (2) search center: GPS coordinate of bookmarked location.

Thus, the Google Ajax APIs are again leveraged to provide directions to and from bookmarked locations using the GLocalSearch, JSONParser and ItemFound classes described in the Search Nearby chapter. The following figure shows the bookmark directions service in action.

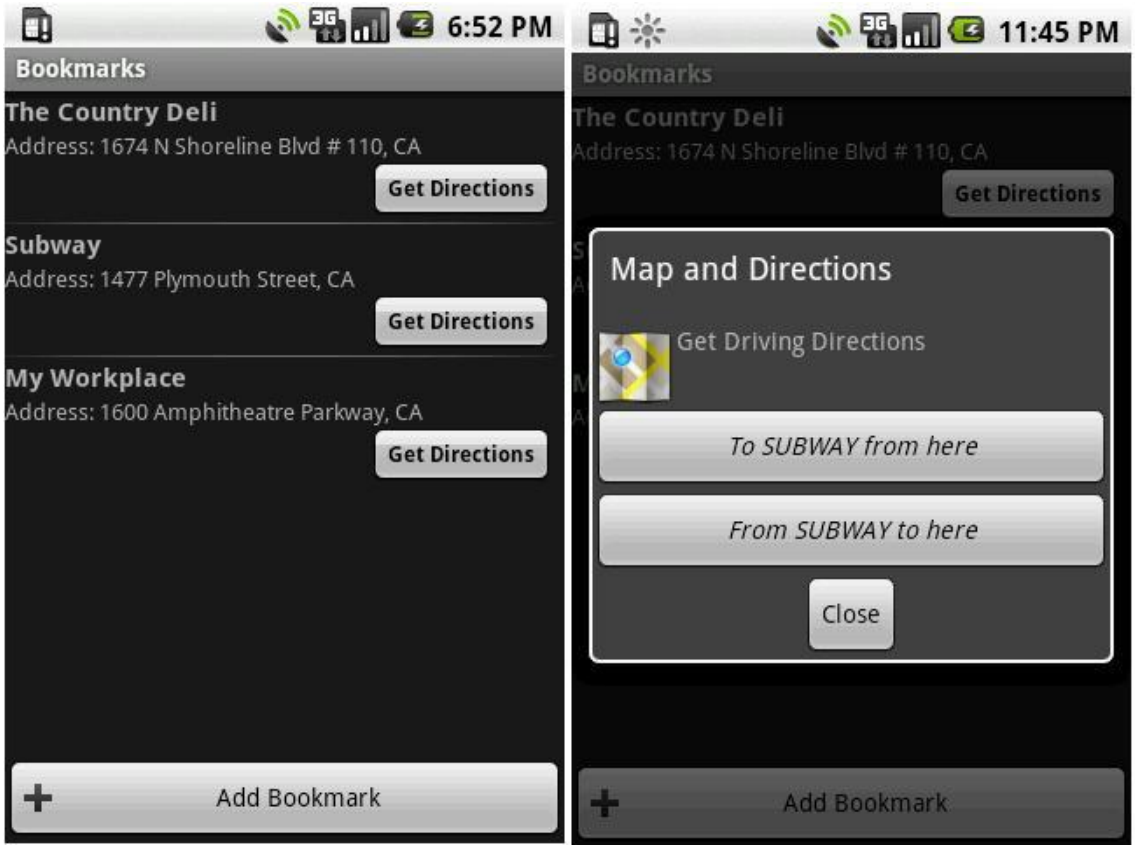

Figure 26: Bookmark directions service

MAP ME

This feature consists of an embedded, interactive Google Map which allows users to view their current location, as well as bookmarks and reminders. The key components of MapMe consist of com.google.android.maps.* library classes such as MapActivity, MapView, MapController, GeoPoint, ItemizedOverlay, and MyLocationOverlay which provide APIs for rendering the map views, zooming and panning, and drawing overlay elements such as: (1) user's current location (2) bookmarks (3) reminders on top of the underlying map view [15].

The MapActivity Class in PlaceMe fetches the GPS coordinates of reminders and bookmarks from their respective SQL databases and creates overlay elements which are associated with “Drawable” resources such as icon files and added to the Map view canvas. The current location of the user is also displayed in real-time as blue dot.

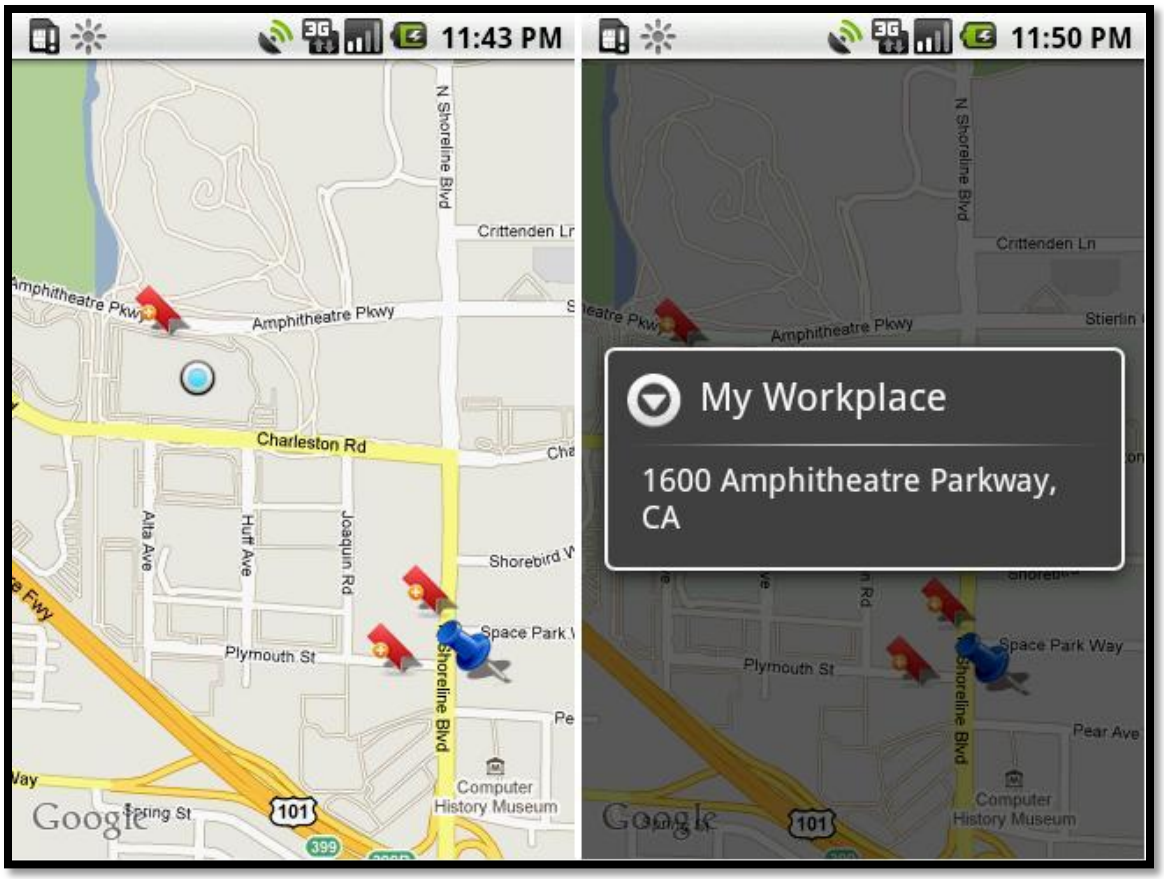

Figure 27: MapMe screen

As shown in the figure, clicking the bookmark icon displays a dialog with the name and address of the bookmark, and in case of the push-pin icon shows the title and address of the reminder for quick reference.

GEO-CODING

When new reminders or bookmarks are created, the user specifies the address of the location but not the GPS coordinates. However, before the reminders or bookmarks are saved to the database, the address entered by the user is translated or geo-coded into GPS coordinates. Geo-coding is performed using the android.location.Geocoder Class [15] which provides APIs for obtaining a set of GPS coordinates matching a given address by performing a network lookup. A set of coordinates is returned since the relationship between GPS coordinates and addresses is not one-to-one. In PlaceMe, we use the first GPS coordinate returned by the geo-coder as the closest approximation.

When the user clicks the "use current address" button in the ReminderEdit or BookmarkEdit screens, PlaceMe obtains the GPS coordinates from the Location Manager and then reverse geo-codes it into an address before saving to database. This reverse geo-coding is also supported by the Geocoder Class [15] which returns a set of matching addresses corresponding to a given GPS coordinate. Again, the first address returned is used as the closest approximation. Aside from bookmarks and reminders, reverse geo-coding is also performed in Search Nearby where the GPS coordinate of the user's location is translated into an address that is displayed below the search box in the "search near: " field to indicates the user's current address.

In order to support easy re-use of geo-coding and reverse geo-coding functionality in various places, we created a GeoCodec class in PlaceMe which provides methods such as getAddressFromGpsPoint() and getGpsPointFromAddress(). This allows greater re-use of code needed to obtain the best matching address or GPS coordinates. In cases when the Geocoder APIs cannot access the network service to perform the appropriate translation or when no matching results are obtained, the address or GPS coordinate to be translated

is used as is where possible. For example, if the user's current GPS coordinate could not be converted to an address, the "search near" text in search nearby would display the GPS coordinates themselves. In cases, where this translation is critical such as when saving a bookmark or reminder, an error dialog is displayed prompting the user to enter a valid address as shown in the figure below.

Figure 28: Invalid address dialog

SURFACE DISTANCE CALCULATION

On many occasions, we needed to compute the distance along the earth's surface between 2 GPS coordinates. The most prominent example is the Reminder Service which uses the distance between the user's current GPS location and that of the reminder to determine whether the user has entered or exited the reminder zone. The simple Great Circle distance (GCD) algorithm provides a reasonably accurate approximation for this purpose, since we're interested in the shortest straight line distance between the two points long the earth's surface [20]. The radius of the earth is taken as 6367 km for all calculations. Essentially, the algorithm translates GPS coordinates from polar to the Cartesian coordinate system, and then computes the great circle distance along the surface of the sphere. The surface distance calculation is also used in search nearby to compute approximate straight line surface distance of different places found from the user's current location to give a rough idea of relative proximity.

For greater re-usability we created the GeoDist class in PlaceMe that provides an implementation of the Great Circle distance algorithm with useful public methods such as getSurfaceDistance() for computing surface distance between 2 GPS points, and inRange() for determining whether 2 GPS points are within a specified range of each other. Having a separate class for this purpose makes it easier to change the underlying implementation of the surface distance algorithm without affecting the rest of the application code and also makes it easier to unit test the code.

Accuracy Analysis

To assess the accuracy of our surface distance calculations, we compared the distances between GPS coordinates of different US cities using our simple GCD

algorithm and a more sophisticated GCD algorithm that uses the Vincenty formula to take into account the flattened shape of the earth [21]. For the tuned GCD which uses the Vincenty formula, the distances were computed using the GPS Visualizer web-based tool [22]. The GPS coordinates of different US cities used as inputs to the algorithm were obtained from "Altas of Texas" [23].

Cities	Tuned GCD (miles)	Simple GCD (miles)
Atlanta-Austin	807.27	805.46
Boston-Dallas	1561.29	1557.94
New York-Fort Worth	1398.12	1395.04
Seattle-Houston	1903.68	1901.32

Table 1: Great Circle distances between major US cities

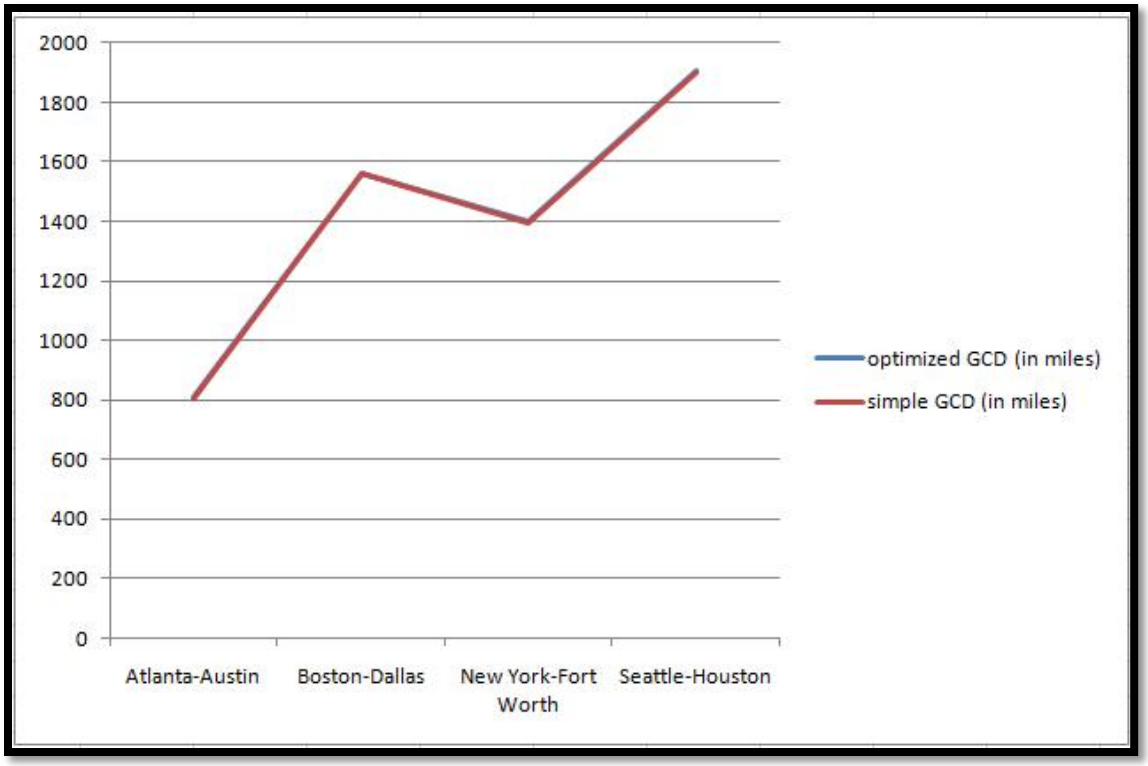

Figure 29: Great Circle distance accuracy comparison

As seen in the chart above, our simple GCD algorithm tracks fairly well to the tuned GCD and produces reasonably accurate results.

Chapter 6: Implementation Notes

This section discusses some of the key aspects of the implementation process such as the development environment used, development timeline, code size, and our solutions to frequently encountered issues in the implementation of PlaceMe app.

DEVELOPMENT ENVIRONMENT

We used Google Code, a web-based application, for project hosting and SVN version control [24]. The integrated issue tracker was used for tracking and managing feature requests and bugs during the development process. The source code for PlaceMe app is available under the terms of the GNU v3 public license at: http://code.google.com/p/geo-reminder/.

Our primary development platform consisted of the following software and hardware configuration:

- **Android SDK**: The Android SDK provides a set of integrated development and testing tools including core Android libraries, a built in emulator, debugger, logger, and device drivers to allow running applications on an Android phone. We used the Google API add-on libraries for implementing some of features in PlaceMe such as geo-coding, mapping, etc.
- **Eclipse Plug-ins**: The Eclipse IDE v3.4.2 was used with the Android Development (ADT) and Subversion plug-ins. The ADT plug-in allows developing and running apps from within the Eclipse IDE and provides access to various features of the Android SDK. The Subversion plug-in was used for working with the SVN repository from within Eclipse IDE.

- **PC**: Windows Vista Home Edition 32-bit, Java Development Kit v1.6.0, 2 GB RAM, 1.5 GHz Intel Centrino dual-core processor
- **Android Phone**: HTC MyTouch 3G [25] with 1.6 Android OS.

TIMELINE & CODE SIZE

This section captures the development effort and timeline for implementing the key features of the Place Me app. Code size is measured in lines of code using the SLOCCount tool [26] and broken down based on feature set. The line counts in the table below only include the Java source code, not the XML view layout or resource files. Timeline to implement the features is estimated based on the actual development time obtained from the SVN log history.

Feature set	Java Classes	SLOC	Timeline (weeks)
Location Reminders	ReminderList.java, ReminderEdit.java, ReminderService.java, RemSvcAutoStarter.java, ReminderDbAdapter.java, Reminder.java	1205	2
Bookmark Places	BookmarkList.java, BookmarkEdit.java, BookmarkDbAdapter.java, Bookmark.java	822	0.5
Map Me	MapMe.java	249	1
Search Nearby	SearchNearby.java	385	1

Table 2 (continued)

Home	Home.java	41	0.1
Library Classes	GeoDist.java, GpsPoint.java, JSONParser.java, GeoCodec.java, GlocalSearch.java, ItemFound.java	553	1
Total		3255	5.6

Table 2: Feature-set timeline & code size

COMMON ISSUES

Here we briefly discuss our solutions to some of the most frequently encountered problems in the context of developing PlaceMe.

SQL memory leaks:

One of the most commonly encountered errors was SQL memory leak which would usually result in crashing the application. Typically these would occur if

references to SQL databases are not properly opened or closed when transitioning between Activities. In order to properly manage database references, our solution was to handle them as part of the Activity life cycle: (1) close all open references in the onPause() or onDestroy() callback methods which are executed before the Activity is paused or destroyed, and (2) open new references when the Activity is resumed or re-initialized in the onResume() or onCreate() methods respectively.

Null-pointer errors:

Null-pointer exceptions usually occur when null in used in cases where an object is expected. The built-in debugger was used extensively to step through the code and root-cause these types of errors in PlaceMe. For example, in the parsing of the JSON search results, field values can sometimes be non-existent. Our solution was to modify the JSONParser class to first check for the existence of the specified JSON object or field before attempting to read it. Another instance was processing of required user inputs when new reminders are created. To prevent null-pointer exceptions, we used defaults to initialize required fields such as “alert type” or “range”. This also circumvented the need to create annoying prompts to force the user to enter a value.

Abstract data typing:

On many occasions, we needed to pass data which contained more than a single primitive type, such as for example, GPS coordinates consisting of latitude and longitude pairs, database row entries consisting of several column fields, and search results consisting of several attributes. Our initial approach was to pass this data in as function

arguments. However, this resulted in the creation of several overloaded functions, one for each possible subset of data that needed to be passed in, unnecessarily bloating the code. To handle these cases, we created abstract data types for aggregating data and associated operations into a Java objects. Some examples include: GpsPoint.java class for GPS coordinates, Reminder.java and Bookmark.java classes for rows in the Reminder and Bookmark tables respectively, and ItemFound.java class for JSON search results. In addition to improved data encapsulation, this also helped simplify method signatures. For example, the methods in the database adapter classes which previously required all the fields in the database table to be passed in order, now only needed a single parameter, namely, the row object, thus minimizing the amount of code changes needed to add new columns to the table. Several other specific issues are also addressed in the subsequent sections on testing and evaluation, and performance enhancements.

TESTING & EVALUATION

This section provides an overview of the methodologies used for testing and debugging of some of the key features and components of the Place Me application, issues encountered and lessons learnt. Key performance enhancements are then described, followed by a comparative evaluation of the Android platform across a set of standard micro-benchmarks.

Chapter 7: Testing

The focus of our test strategy was primarily functional and end-to-end testing due to the limited development time frame, novelty of the platform and the sheer variety and number of components involved. The Android SDK provides a set of integrated development and testing tools that include a built in emulator, debugger, logger, and device drivers that allow running applications on an Android phone. The emulator was used extensively in the initial development and testing of PlaceMe.

TESTING USER INTERFACES

User interfaces in Android are typically specified as XML layout files consisting of parameterized View objects instantiated at run-time. Similar to HTML tags, the names of XML elements and attributes correspond to that of View classes and instance variables in the Android framework.

Most of the design and testing of UIs in PlaceMe was done using the DroidDraw tool, a 3rd party view editor that runs as a standalone Java executable [27]. It allows creating views graphically and generates the underlying XML code. Rather than use the

tool for creating views, we used the tool more for validation of our XML layouts. It allowed quickly tweaking View objects and parameters in XML and observing the resultant output, ideal for iterative code development. In addition, since the View objects are instantiated at run-time, the Android emulator itself allows hot-swapping the XML layout files without re-installing and re-running the entire application, provided the underlying Java code is not changed. On occasions, this method was used for testing small changes in UI layout.

DATABASE TESTING

PlaceMe uses SQL databases for managing reminders and bookmarks. In order to test the basic Create, Read, Update, Delete (CRUD) operations on these databases we used the built-in Android Debug Bridge (ADB) tool [28] which is part of the Android SDK. The ADB tool is client-server program that allows communicating with the emulator or device over TCP network ports via a client-side shell. This makes it possible to examine state of the database from a remote shell using the sqlite3 command line tool and execute SQL commands [29]. The figure below shows the sqlite3 tool in action when used to print out all entries in the bookmarks table in PlaceMe app.

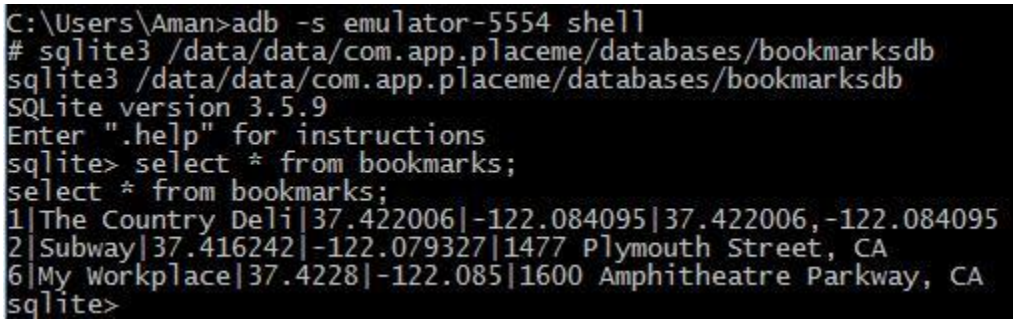

Figure 30: Using Sqlite3 tool to examine databases

We used the ADB tool to visually verify the result of executing SQL write operations to add entries to database tables using the ReminderDbAdapter and BookmarkDbAdpater Classes. On occasions we encountered unexpected SQL read/write errors when the database schema was modified by adding, editing or deleting column names in the tables. The ability to visually verify the table entries allowed us to root cause the issue. We realized that when the database schema is updated, any pre-existing entries are left in an inconsistent state. As a result future read/write operations result in run-time SQL exceptions. The solution was to erase all data and re-create the database tables when the schema is modified.

STRESS TESTING

The Android platform provides a built-in Monkey tool [29] which runs on the emulator or device and allows sending pseudo-random sequences of events to the application via the ADB shell. We used this tool for stress testing PlaceMe by sending 1000 sequences of random events to it which include button presses, random data entries, scrolling, screen transitions, navigation between Activities, etc. The figure below shows a screenshot of a partial command line execution of the Monkey tool on PlaceMe.

Figure 31: Using Monkey tool to generate pseudo-random events

We used the Monkey tool to stress test PlaceMe on both the emulator and an actual phone. We found no unexpected application crashes, unhandled exceptions or "application not responding" errors. All operations were successfully completed including persisting reminders and bookmarks to the database.

SIMULATING MOBILITY

Since Place Me is a location-based application, the ability to simulate mobility is central to testing most of its core features. The Android SDK provides Dalvik Debug Monitor Server (DDMS) tools [29] which connect to the ADB service described above and provides a communication bridge between the Eclipse IDE and the emulator or phone. As shown in the figure below, the DDMS Perspective in Eclipse provides Location Controls which allow sending location coordinates to the emulator manually, or playback/stream the GPS route information at varying speeds using GPX or KML file [29].

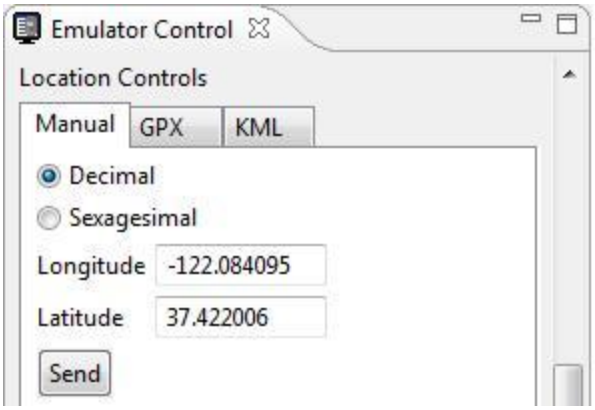

Figure 32: DDMS location controls

We used this extensively for testing the ReminderService, MapMe and Search Nearby Activities in PlaceMe by sending mock location coordinates while the Activity under test is running on the emulator. In case of the ReminderService this allowed us to discover and resolve bugs in the tracking algorithm that triggers reminders based on entry/exit from reminder zones. The route playback feature also allowed us to test the responsiveness of the ReminderService due to frequent location changes at high mobility rates, such as when the user is driving. In case of MapMe, this allowed us to test the auto-scroll and zoom-settings which update the map view in response to changes in user's current location. Finally, in Search Nearby this allowed us to simulate and test how the user's current location address is updated in response to location changes.

UNIT TESTING

We independently unit tested and validated the supporting classes in com.placeme.lib package, which are designed to provide re-usable functionality used across different features such as building and issuing Google Ajax queries (GLocalSearch), parsing the JSON results (JSONParser), calculating surface distance (GeoDist), and geo-coding (GeoCodec). For example, the GLocalSearch and JSONParser classes, were tested as a standalone Java console application using JUnit test cases.

FIELD TESTING

PlaceMe was field tested on, MyTouch 3G [25], an Android powered mobile phone with Android 1.6 OS. Using the app on an actual handset in real life scenarios revealed some interesting insights which would have been otherwise difficult to detect in a simulated environment. For example, initially we used meters, instead of miles to specify the radius of the reminder zone. During actual usage, we found that reminders would not trigger when they were supposed to. At first this appeared strange, since a thorough re-testing of the tracking algorithm in the reminder service did not reveal any issues. We realized later the reason the reminder alerts were not triggering was because the reminder zones were never really being entered or exited. Some of the issues involved were: (1) geo-coding a US street address does not always yield a unique GPS coordinate (2) the GPS coordinate returned can be off by several meters from the physical address (3) meters is too fine grained for defining reminder zones. This led us to use miles instead, and set the default reminder radius to a quarter mile, in order to compensate for the crudeness inherent in geo-coding. According to Hartnett's article [30], the Google Geo-coder can be off by 509 feet on average or 155.14 meters. Moreover, miles is a more

natural unit of measurement for this purpose as most Americans tend to think of distances in miles rather than meters.

Some of the other insights we gained was the importance of reducing manual typing of addresses where possible to reduce data entry errors due to the limited screen real-estate and keyboard size. This motivated the addition of automated address entry features like: (1) “use current address” and “select from bookmark” buttons when creating new reminders or bookmarks (2) ability to bookmark search results using a “bookmark” button in Search Nearby. In addition, some of the performance enhancements, discussed in the next section, were first discovered during field testing.

Chapter 8: Performance Enhancements

In this chapter we briefly discuss some of the key performance enhancements and feature improvements made in the course of developing and testing the PlaceMe application.

THREADING EXPENSIVE OPERATIONS

Time or resource intensive operations such as geo-coding and reverse geo-coding, writing to database, Ajax API calls are best handled in a separate thread than the primary UI thread. This prevents blocking the main thread that renders the UI and keeps it from becoming unresponsive while these operations are being performed. Consistent with Android best practices, we spawn off separate worker threads to perform the expensive operations, which then post the results back to the parent UI thread and update the views accordingly. Some of the key areas where expensive operations are threaded in Place Me are discussed below.

As discussed earlier, before new bookmarks or reminders created by the user are persisted to the database, the address is geo-coded into GPS coordinates (or reverse geo-coded if “use current address” button is used) using Geo-coder APIs which perform network look-up. If the operation is successful the entry is written to database. Here the geo-coding and database write operation is threaded. As shown the figure below, a progress bar screen is displayed during the process.

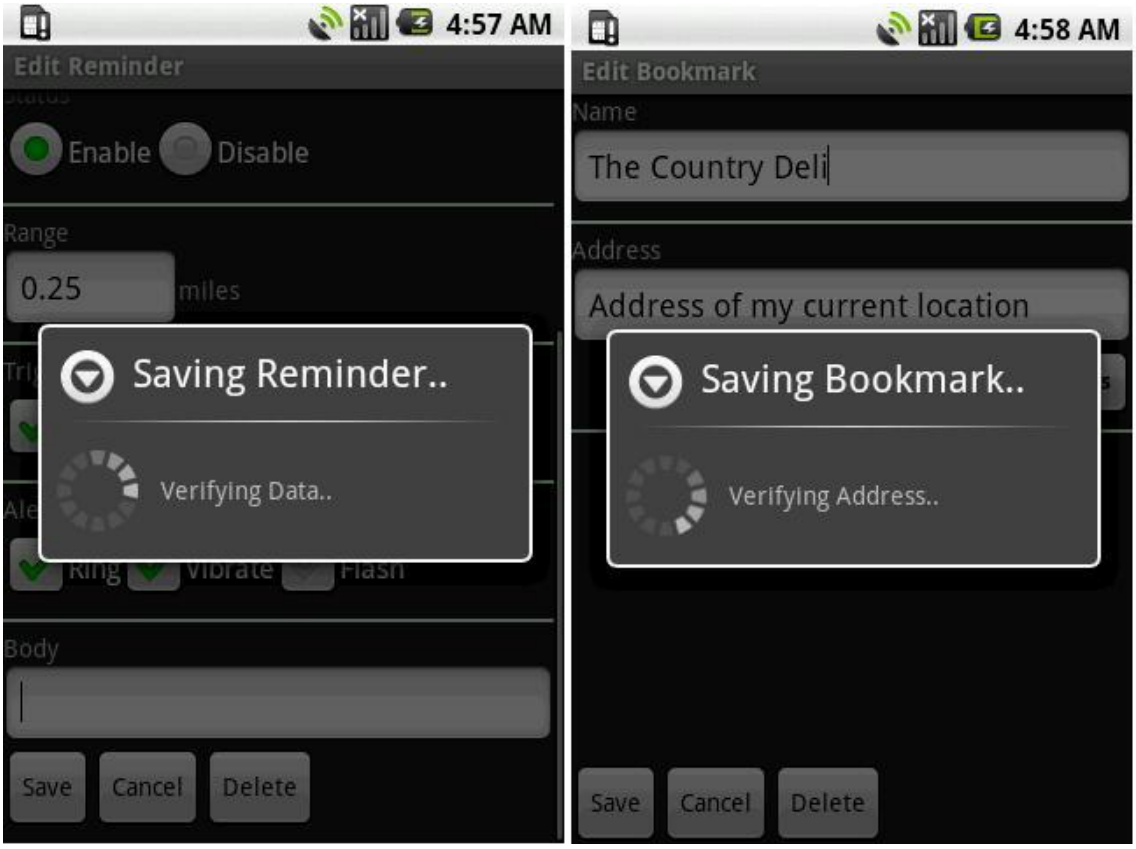

Figure 33: Progress bar when saving reminders (left) and bookmarks (right)

Another example is the Search Nearby feature. When the Activity is opened, the user's current GPS location is reverse geo-coded into an address which is displayed below the search box. The GPS to address translation is performed in the separate thread, and the view is updated with the address found. A momentary progress bar dialog is displayed while this occurs.

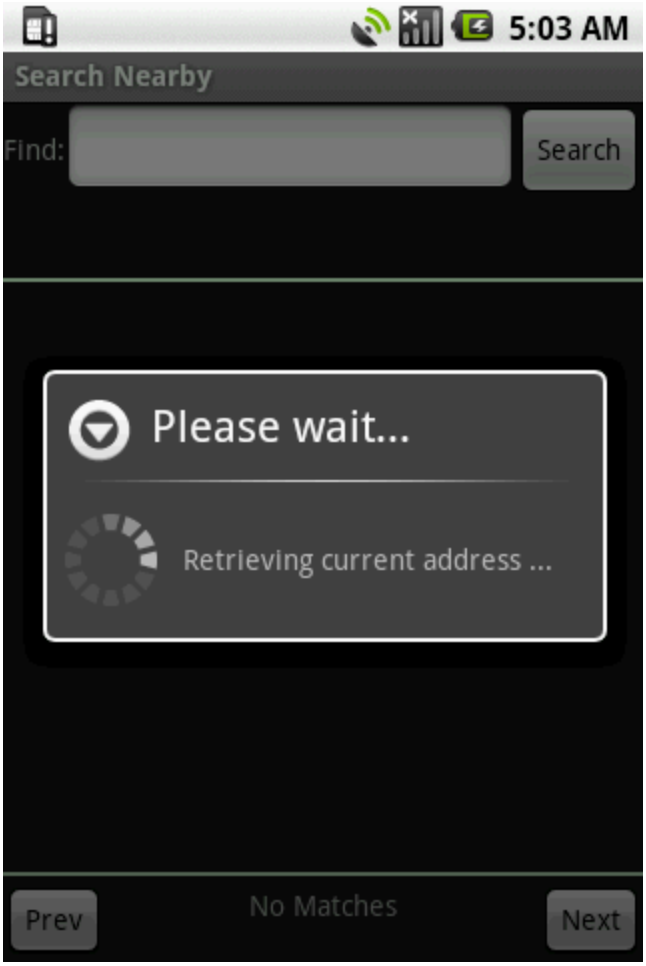

Figure 34: Address retrieval progress bar in Search nearby

The “search” button in Search Nearby screen performs a network query using Ajax APIs to obtain the local search results from Google servers. In addition, every time the “prev” and “next” buttons are used to fetch the previous or next batch of search results, respectively, another network query is performed. Here the issuing of the network query and parsing of JSON results is done in a separate thread.

REAL-TIME LOCATION UPDATES

During real-world testing and evaluation of Place Me app on an actual Android phone, we identified a performance issue in the implementation of Search Nearby, where if the user is mobile (ex: driving or on the bus) while using the feature, the Activity would continually attempt to update the user's location displayed under the search box (as explained in Chapter 15 above). This would cause the progress bar to pop-up repeatedly while the reverse geo-coding is performed, in many cases, causing the application to crash. The crux of the problem was to update and display the user's current location in timely manner while minimizing continual location updates that cause the app to crash. Our solution was to essentially combine the location update and the fetching of new search results in a single operation. This means that every time the user hits the "search", "prev" or "next" buttons 2 things would occur: (1) search results are updated (2) the user's location is refreshed/updated. This made it possible to keep the user's location as "current" as possible without continually updating it, thus saving repetitive network queries (needed for reverse geo-coding) and making the feature usable. Thus, only when the search results are updated, would we update the "search near" text to reflect the user's present location.

Most of the Activities and Services in PlaceMe keep track of the user's current location, namely, Reminder Edit, Bookmark Edit, Map Me, Reminder Service, etc. The android.location.LocationManager Class [15] is used to poll for the GPS information. In the implementation of PlaceMe, we control the frequency of GPS polling to conserve power and extend battery life where possible. For example, in the Reminder Service we register to receive GPS location updates only if the device moves by more than 100 meters. This minimizes idle polling when the user is stationary (which in most cases

happens quite frequently such as when people are at work or at home). In other cases, such as the MapMe Activity, which displays the user's current location in real time, we turn off location monitoring when the Activity is stopped, paused, or exited and thus no longer in active view. Location monitoring is resumed when the Activity is brought back into focus and becomes active again. This helps further minimize GPS polling when not needed.

NETWORK I/O & MEMORY USAGE

As discussed earlier, the "directions" feature uses AJAX network queries to fetch the URLs to Google Map directions. To minimize run-time overhead and network bandwidth utilization where possible, these URLs are not obtained and stored beforehand. Instead, they are obtained on-demand at the time when the user requests them. For example, the directions dialog in Bookmark List provides options to get directions to or from the bookmarked place. Only when user actually clicks the button to show directions, is the network query to fetch the URL initiated. In addition to reducing unnecessary network I/O, this also helps provide "real-time" directions, based on the user's present location at the time he requests them.

When using the Search Nearby feature to locate nearby places, users are typically only interested in the top 4-5 search results. However, the Google Local search produces several thousand hits. To minimize network I/O and memory usage, we obtain 8 search results at a time (max. results returned by a single Google AJAX API call) instead of performing several network queries to obtain more results and buffering them in memory. As shown in the figure below, we use "virtual" pages and allow navigating to next (or

previous) set of results. A network query is performed to obtain the next (or previous) set of results, only when the user clicks the “next” or “prev” button (on-demand basis).

Figure 35: Results navigation in Search Nearby

When possible we try to avoid network IO completely. For example, when the “select from bookmark” button is used to populate the address field of a reminder, we do not geo-code the address. Instead the GPS coordinate is read from the bookmarks database. Similarly, when a search nearby result is bookmarked, the GPS coordinate is obtained directly from the JSON object instead of fetching the address and geo-coding it. Reducing network I/O in this manner also helps conserve battery power.

DATABASE QUERYING & FILTERING

The ReminderService periodically sifts through the reminder database to determine when to trigger reminder alerts. To help reduce the search time, we use SQL WHERE clause to filter out the “enabled” reminders and only iterate through them. As an added optimization feature, we can potentially also use the WHERE clause to query for reminders whose reminder zones contain the user’s current GPS location, in order to further reduce the search size.

Furthermore, to reduce the amount data that needs to be analyzed, we use database cursors to obtain only the relevant fields needed for each entry. Passing the database cursor between different functions in the Reminder Service is more memory

efficient than passing the actual row data. Database cursors are conceptually analogous to pointers in C++, except that they point to rows of a database table instead of data types.

Chapter 9: Micro-Benchmarking

In order to assess and characterize the computing power and performance of the Android framework, we performed experiments to measure and evaluate the execution time for a series of targeted benchmark tests. We expect a lot of developers who are using the platform or are adopting it for the first time, would be interested in these results as Android gains greater traction in the developer community and the app-market.

For comparison purposes, we ran our benchmarks against 3 different platforms: (1) Android emulator (2) Android phone (3) PC. The PC provided a good baseline for comparing the benchmark data and served as a frame of reference for analyzing the results. In this section we briefly describe our benchmarking framework, discuss the design of the benchmark tests and compare the benchmark test results across all 3 platforms.

BENCHMARK FRAMEWORK

Our benchmark framework is a set of classes designed to support the creation, execution and measurement of targeted benchmark tests. It comprises of the following key elements:

- StopWatch class that allows timing the execution of a particular benchmark test

- TestUtils class used for generating random test inputs used for running the benchmark tests, and provide other common utilities such as logging test info, type conversions, etc.
- Stats class for storing, formatting and printing the result summary of benchmark test execution
- A set of classes that implement the benchmark tests
- BenchmarkSuite class for instantiating the tests with different input parameter sizes and timing their execution
- TestRunner classes for running the benchmarks on the PC, emulator and phone.

The figure below shows a UML class diagram of the benchmark framework. In order to support running benchmark tests on the PC, emulator and phone, we created 2 separate project configurations: (1) one for running the tests as a standalone Java console application on the PC (2) another for installing and running the test as a simple Android application on the emulator and phone. Due to the differences in the underlying platform infrastructure between these 2 configurations, we had to modify our implementation of some of the benchmark tests to work around the limitations of each. For example, one of our benchmark tests consisted of drawing random rectangles on the screen. On the PC this can be accomplished by using the AWT and Swing libraries in Java. However, Android implements its own UI framework and does not support these libraries. As a result, we had to implement the same benchmark test for Android using its native UI framework and view objects. The next section discusses the benchmark tests in more detail.

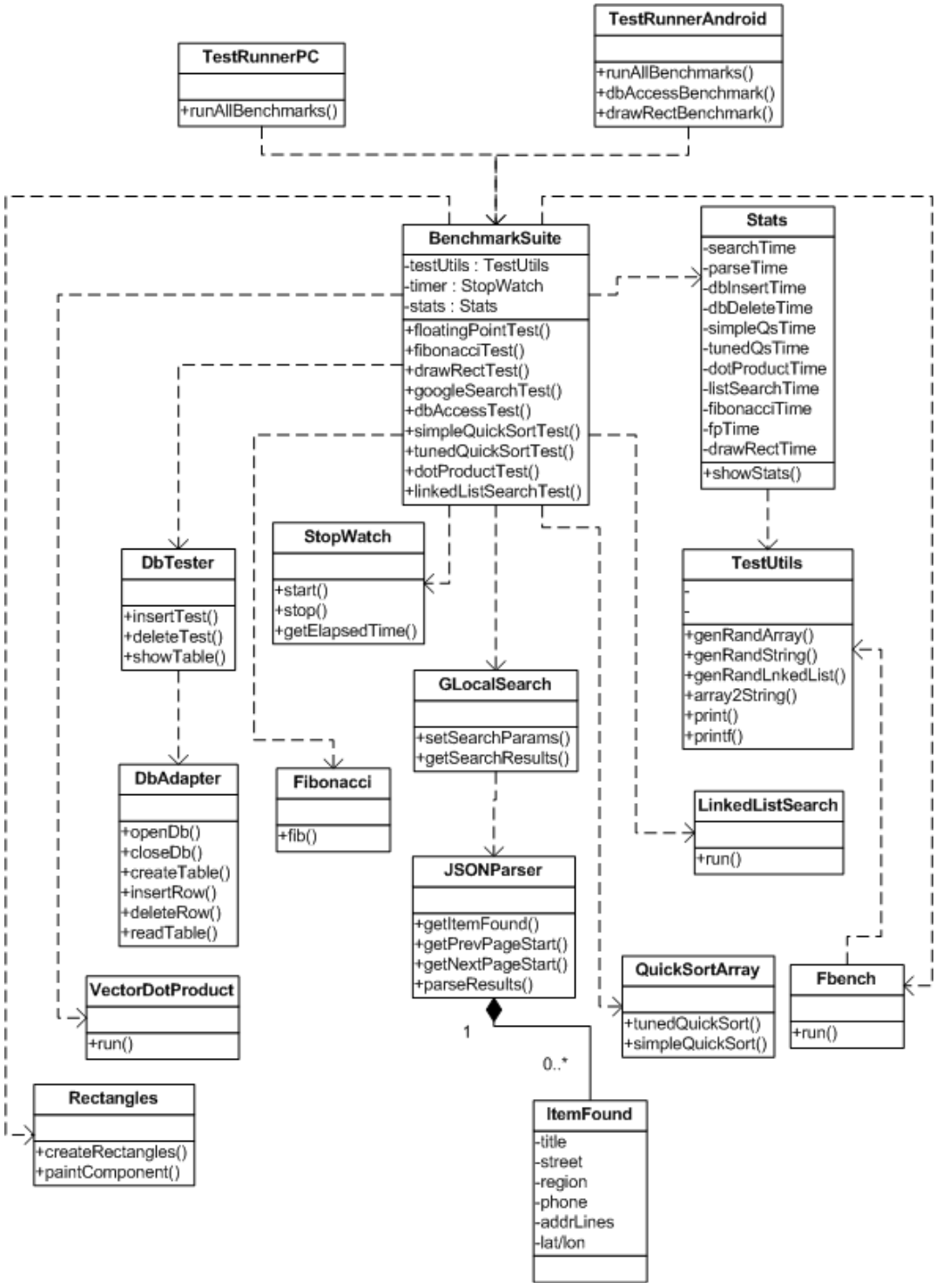

Figure 36: Benchmark infrastructure class diagram

BENCHMARK TESTS

Here we briefly describe the tests used for micro-benchmarking the Android platform. These tests were designed to obtain measurements of the raw computing power of platform such as the ability to perform floating point operations, vector dot products, etc., as well as some of the core infrastructure elements used in the PlaceMe app such as network I/O, databases and graphics.

Quick-sort

This test computes the time to sort a randomly generated array of integers using the quick-sort algorithm. Our goal here is to evaluate the performance of the system with regard to sorting efficiency. We used 2 flavors of the quick sort algorithm: (1) simple quick sort algorithm (2) tuned Quick-sort provided by sort method of the Java Array Collections class which grows the stack as O(log n) instead of O(n) and handles duplicates to produce balanced splits.

Vector Dot-Product

This test computes the time to perform the dot-product of 2 randomly generated arrays of doubles. Our objective here is to determine the computational speed of the system. We have used doubles data type since it is larger and thus more computationally intensive to multiply than integers or floats.

Linked List Search

This test computes the time to search through a linked list of randomly generated strings. The idea here is to assess and measure the ability of the system to iterate and search through large data structures. String search in a linked list data structure provides a reasonably good test for this purpose, as it entails several accesses to possibly non-contiguous areas of memory. We used randomly generated, 128-bit UUID strings, consisting of alpha-numeric characters, as the search data in the linked list.

Fibonacci

This test measures the time to compute the Fibonacci number using a recursive implementation. The idea here is to measure how fast the system can respond to several function calls that consume memory stack space.

Floating Point Test

This test measures the average time to run a single iteration of FBench, a suite of trigonometric floating point benchmark tests [31], which exercise the Floating-point unit or FPU. The Fbench test runs an implementation of the "optical raytracing algorithm" that is particularly sensitive to errors, on a real world application and measures its accuracy. Our goal here is to assess the ability of the system to perform complex floating point computations accurately.

Issuing Search Query

This test calculates the average time to retrieve results from a randomly selected Google local search query. The goal here is to measure the network data access speed for performing common Google search queries. A random query string is selected from a dictionary of pre-defined search queries which are known to return non-zero search results. We used the following common search terms for the dictionary of search queries (also provided on the pop-up menu in the Search nearby screen in PlaceMe): “pizza”, “coffee”, “gas”, “restaurant”. One these search terms is randomly selected per test iteration. The GLocalSearch class we created for PlaceMe was used for creating and issuing the Google search queries.

Parsing Search Results

While the above test measured the time to retrieve search results from Google, this test computes the average time to parse the JSON search results returned from a single Google local search query. The idea here is to determine how fast the system can parse the JSON object data and populate the results into native Java objects. The JSONParser class we created for PlaceMe was used for parsing the search results.

Database Access

This test computes the average time to insert a randomly generated entry into a database table, as well as the average time to delete an entry. Our goal here is to measure the database access efficiency with regard to insert and delete operations. The test creates a database table, and inserts several entries into it. Subsequently, it deletes a random number of rows.

In order to run benchmark database access tests on the PC, we had to create a database on the PC. Our first approach was to use Android's native SQL drivers, however, we soon realized that this would not be feasible as it is deeply integrated into the framework and has dependencies on core Android classes which are instantiated behind the scenes at run-time and thus cannot be easily mocked. Our final solution was to create a standalone SQL database implemented using the Java JDBC libraries and the SQLIteJDBC driver v056 (based on SQLite 3.6.14.2) for running the database benchmark tests on the PC [32]. We created a DbAdapter class to provide an application level database abstraction layer for allowing clients to setup and interact with the database tables. A separate DbTester class, we created, was used to implement the insertion and deletion tests which used the DbAdapter class for database access.

Drawing Rectangles

This test computes the time to draw random rectangles on the screen. Our objective here was to measure the efficiency of the system with regard to creating and rendering simple graphical objects on the screen. The test consisted of drawing rectangles of random dimensions (length/width) and a randomly selected X-Y coordinates within the view Frame.

Similar to the database benchmark tests we had to create separate tests for emulator/phone and the PC since the Java AWT and Swing graphics libraries are not supported in Android as mentioned earlier. In order to draw random rectangles on the screen in Android we extended the View object and used the Shape class to define the dimensions of the rectangles.

TEST PARAMETERS & INPUT SIZES

The test parameters and input sizes used for each of the above benchmark tests are shown in the table below:

Test	Parameter	Input Size
Simple QuickSort	array size	50000
Tuned QuickSort	array size	50000
Vector Dot Product	vector size	50000
Linked List Search	linked list	50000
Fibonacci	N	30
Floating Point Tests	# iterations	50000
Issuing Search Query Parsing Query Results	# queries	10
Database Access	# insertions	100
	# deletions	100
Drawing Rectangles	# rectangles	1000

Table 3: Benchmark Test Parameters & Sizes

TEST RESULTS

Here we present the results of running the benchmark test with the above input sizes on the following platform configurations:

1. Emulator, 1.6 API Level, Google API Add-on
2. Phone: MyTouch 3G, Flash: 512MB, RAM: 192MB, 1.6 Android OS
3. PC: 2GB memory, 1.50GHz processor, JDK 1.6, Windows Vista, Eclipse v.3.4

The table below shows the execution times for each benchmark test (all test times are in milliseconds).

Test	run time (ms)		
	emulator	**device**	**pc**
avg. search query	2.4000	1.0000	3.0000
avg. query parse	805.7000	785.8000	158.6000
avg. db row insertion	161.7700	30.3400	152.3600
avg. db row deletion	91.9900	26.0000	133.2500
linked list search	177.0000	160.0000	4.0000
vector dot product	44.0000	22.0000	6.0000
simple qsort	447.0000	205.0000	7.0000
tuned qsort	491.0000	218.0000	18.0000
Fibonacci	4527.0000	2007.0000	26.0000
rectangle draw	1100.0000	692.0000	256.0000
floating point calc	2.4703	0.8653	0.0104

Table 4: Benchmark Test Results

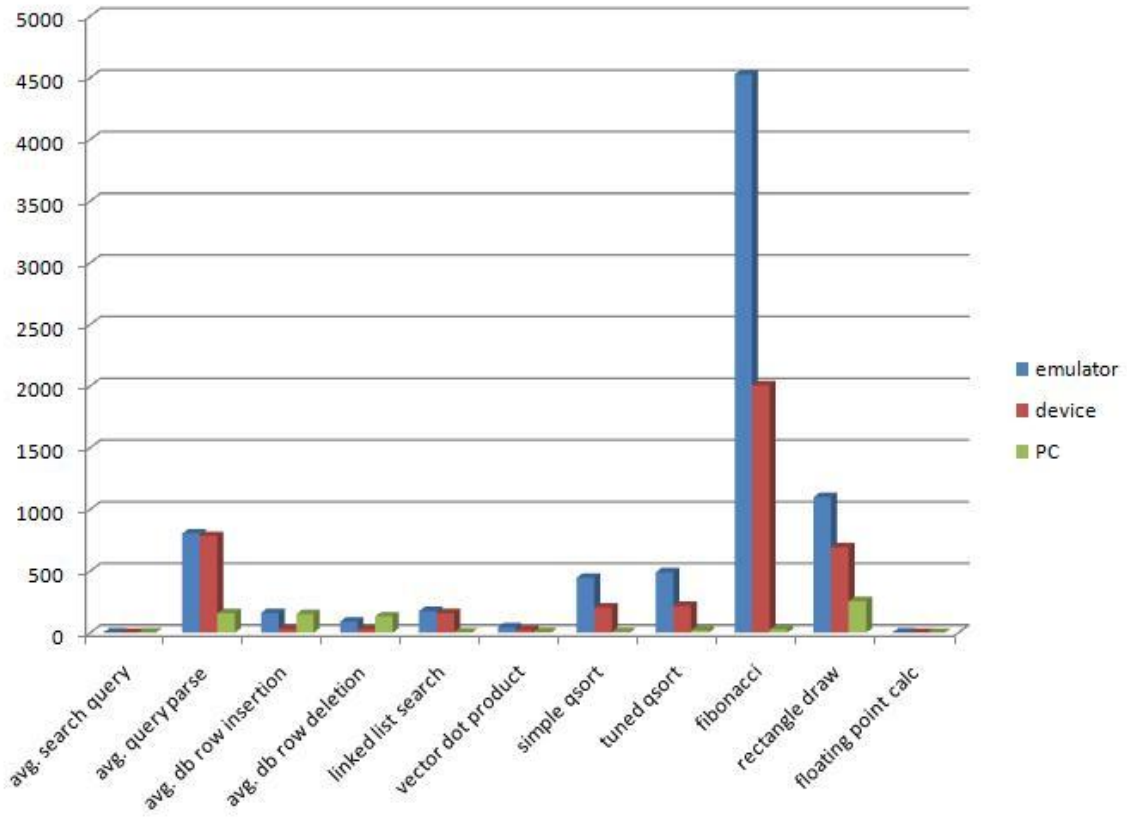

Figure 37: Benchmark Test Time Comparison

The graph above shows the relative performance of the PC, emulator and phone across all benchmarks. As expected, for almost all of the benchmark tests, the PC's performance was orders of magnitude faster (due to more powerful hardware), followed by the phone and then the emulator. Surprisingly, the phone outperformed the PC in database access benchmarks. We suspect that Android platform might be loading the entire database in memory for highly efficient access. In addition, Android's native SQLite run-time libraries are probably optimized for performance on the Dalvik virtual machine.

The following figure shows the output of running the benchmark suite on the PC.

```
-------------------- Benchmark Summary ----------------------
Google AJAX Search Query     : 30
Google AJAX Result Parse     : 1586
Database row insertion       : 15236
Database row deletion        : 13325
Random Linked list Search    : 4
Random Vector Dot Product    : 6
Simple Array QuickSort       : 7
Tuned Array Quicksort        : 18
Fibonacci Recursive          : 26
Draw Random Rectangles       : 256
Floating Point Calculations: 0.0104
-------------------------------------------------------------
```

Figure 37: Benchmark test output on PC

The figures below show the screenshots of the BenchmarkApp we created to perform benchmark testing on the emulator (left) and the phone (phone). The test times for each benchmark are printed to the screen. The green shaded area at the bottom shows a frame layout where the 1000 random rectangles are rendered as part of the "draw rectangle" benchmark test.

Figure 38: Benchmark test output on emulator (left) and phone (right)

DISCUSSION

In this report we have presented the design, implementation and evaluation of PlaceMe, an Android app that provides location based services such as location reminders, bookmarking places, mapping and search nearby. Here we discuss some future development ideas and conclusions.

Future Work

The PlaceMe app can be improved and extended in several ways. Here we explore some of these exciting possibilities and discuss our future plans for adding new features that build upon and compliment the current functionality.

In addition to location, time can be added as another contextual element to the location reminders. This will allow users specify a time and calendar date when location is not sufficient to define the context in which the reminder should be received, making it possible to setup recurring reminders. In addition, a reminder snooze feature which allows temporarily suppressing reminder alert notifications would be ideal. Moreover, we would also like to integrate the PlaceMe reminder system with the built-in Calendar app for easy access and allow synchronization with web-calendars such as Google Calendar.

Social networking features can help leverage the location based services, adding a whole new dimension of possibilities. For example, sharing of bookmarked locations between Facebook friends would make it easy for people to exchange addresses of popular restaurants, tourist attractions, hang-out destinations, etc. Bookmarks can also be extended to include geo-tagged photos, videos, reviews and articles. In addition, MapMe can be updated to show nearby friends on the map in real-time and send alerts when they are immediate vicinity, making it easy for people to connect with each other.

Location centric micro-blogging is another feature we would like to explore. This would allow users to blog on websites such as Twitter, in real-time, directly from their phone and share location specific information such as photos and videos of the places they visit or publish reviews of restaurants, shops, etc.

Finally, we would like to create a server side application to provide services such as allowing users to track their friends' location in real time, provide personalized place recommendations based on the user's past history of visited places such as department stores, restaurants, shops, etc, and allowing users to review or rate places they visit and share their experiences with other users of PlaceMe app. This would entail communicating with server side protocols to send and receive location specific information.

Conclusion

Mobile phone users today have a plethora of apps available at their fingertips, from text messaging and social networking to real-time video streaming and immersive 3-D gaming. A multitude of commercial and personal apps have been developed for popular phone platforms such as Android and iPhone. Thriving developer communities have evolved to support the growing user-base, which continue to make available thousands of new apps to mobile phone users. The pervasiveness of mobile phones and sheer size of the app market has also opened new avenues for targeted marketing and advertising campaigns that provide businesses greater visibility and access to end-users.

As mobile handheld devices continue to evolve, they will capable of supporting increasingly sophisticated and feature rich applications, which truly have the potential of making a profound impact in people's everyday lives and changing the dynamics of social networking, communication and mobile entertainment. The possibilities are endless and we have only just begun to scratch the surface.

REFERENCES

[1] U. Government. (1999) Global Positioning System. [Online]. www.gps.gov

[2] Wi-Fi.org. (2010) Wi-Fi Alliance. [Online]. http://www.wi-fi.org/discover_and_learn.php

[3] Federal Communications Commission. (2002, Nov.) Third Generation Wireless Systems. [Online]. http://www.fcc.gov/3G/#sec2

[4] HowStuffWorks.com. (2001, Sep.) How do touch-screen monitors know where you're touching?. [Online]. http://computer.howstuffworks.com/question716.htm

[5] Bluetooth Special Interest Group. Bluetooth.org. [Online]. https://www.bluetooth.org/apps/content/

[6] Wikipedia.org. (2010, Oct.) Accelerometer. [Online]. http://en.wikipedia.org/wiki/Accelerometer

[7] D. Rusling, *The Linux Kernel*. Berkshire, United Kingdom, 1996. [Online]. "http://tldp.org/LDP/tlk/tlk.html"

[8] Oracle.com. (2010) Java . [Online]. http://www.oracle.com/technetwork/java/index.html

[9] Android Feeder. [Online]. http://androidfeeder.com/

[10] T. Foursquare. (2010) FourSqaure. [Online]. http://foursquare.com/

[11] (2008) Loopt. [Online]. http://www.loopt.com/

[12] G. Inc. (2010) Gowalla. [Online]. http://gowalla.com

[13] T. Sohn, et al., "Place-Its: A Study of Location-Based Reminders," in *UbiComp*, 2005, p. 19.

[14] H. Oinas-Kukkonen, V. Kurkela, and I. Oulu, "Developing Successful Mobile Application," in , 2003, p. 5.

[15] Google Inc. (2007, Nov.) Android API Reference.

[16] G. Inc. (2010, Nov.) Android Developer Guide. [Online]. http://developer.android.com/guide/basics/what-is-android.html

[17] A. D. Team. (2007, Nov.) YouTube presentation. [Online]. http://www.youtube.com/watch?v=Mm6Ju0xhUW8

[18] S. Hashimi, S. Komatineni, and D. MacLean, "Pro Android 2," in *Pro Android 2*. New York, US: Apress, 2010, ch. 1, pp. 10-11.

[19] Google Inc. (2010) Google Code Web Search API. [Online]. http://code.google.com/apis/ajaxsearch/documentation/reference.html#_fonje_local

[20] Carroll College. (1996) Spherical Coordinates & GPS. [Online]. http://www.math.montana.edu/frankw/ccp/cases/Global-Positioning/spherical-

coordinates/learn.htm

[21] Movable Type Ltd. Vincenty formula for distance between two Latitude/Longitude points. [Online]. http://www.movable-type.co.uk/scripts/latlong-vincenty.html

[22] A. Schneider. (2003-2010) GPSVisualizer. [Online]. http://www.gpsvisualizer.com/calculators#distance

[23] University of Texas at Austin. (2006, May) Web Library, Altas of Texas. [Online]. http://www.lib.utexas.edu/maps/atlas_texas/great_circle_distances.jpg

[24] Google Inc. Google Code Project Hosting. [Online]. http://code.google.com/

[25] T-mobile USA, Inc. (2002-2010) T-mobile myTouch 3G. [Online]. http://www.t-mobile.com/shop/phones/cell-phone-detail.aspx?cell-phone=MyTouch-3G-White

[26] D. Wheeler. SLOCCount. [Online]. http://www.dwheeler.com/sloccount/

[27] brendan.d.burns. DroidDraw. [Online]. http://www.droiddraw.org/

[28] Google Inc. (2010, Nov.) Android Developers. [Online]. http://developer.android.com/guide/developing/tools/adb.html

[29] Google Inc. (2010, Nov.) Android Developers. [Online]. http://developer.android.com/guide/developing/tools/adb.html#sqlite

[30] W. Hartnett. Google Maps vs. Virtual Earth: A geocoding accuracy showdown in West Palm Beach. [Online]. http://www.wmhartnett.com/2008/01/15/google-maps-vs-virtual-earth-a-geocoding-accuracy-showdown-in-west-palm-beach/

[31] J. Walker. (2007, Oct.) Fbench: Trigonometry Intense Floating Point Benchmark. [Online]. http://www.fourmilab.ch/fbench/fbench.html

[32] Zentus.com. SqliteJDBC. [Online]. http://www.zentus.com/sqlitejdbc/